THE DOS/BATCH OPERATING SYSTEM

PART 3

THE DOS/BATCH MONITOR

PART 3

CHAPTER 1

INTRODUCTION TO THE MONITOR

1.1 THE DOS/BATCH MONITOR

The PDP-11 Disk Operating System (DOS/BATCH) Monitor is a powerful, keyboard and batch-oriented program development system designed for use on PDP-11 computers. The Monitor facilitates use of a wide range of peripherals available for use with the PDP-11.

The Monitor supports the PDP-11 user throughout the development and execution of his program by:

- providing convenient access to system programs and utilities such as the FORTRAN Compiler, the MACRO Assembler, a Linker, a debugging package, an Editor, a file utility package, etc.;
- performing input/output transfers at three different levels, ranging from direct access of device drivers to full formatting capabilities, while providing the convenience of complete device independence;
- providing a file system for management of secondary storage; and
- providing a versatile set of keyboard commands for use in controlling the flow of programs.

System programs and utilities can be called into core from disk, DECtape, magtape, cassette or paper tape with Monitor commands issued directly from batch streams or the keyboard. This feature eliminates the need to manipulate numerous paper tapes, and provides the user with an efficient and convenient programming tool.

DOS/BATCH gives the user program the capability of complete device independence. Programs can be written without concern for specific I/O devices. When the program is run, the user can select the most effective or convenient I/O device available for the function to be performed. In addition, if the system configuration is altered, many programs can take advantage of the new configuration without being rewritten. Logical names can be assigned to devices within the system, enabling symbolic referencing of any device. No concern need be given to I/O buffer size within the user program, yet the user can alternatively retain direct control of I/O buffers.

All input/output (I/O) transfers are handled by the Monitor in any of three user-selected levels called READ/WRITE, RECORD/BLOCK, and TRAN. READ/WRITE is a formatted level of I/O in which the user can specify any one of nine options. RECORD/BLOCK is a file-structured, random-access I/O level with no formatting.

TRAN does basic I/O operations at the device driver level. All I/O is concurrent and interrupt-driven.

The file system on secondary storage uses two types of files: linked and contiguous. Linked files can grow serially and have no logical limit on their size. Contiguous files must have their lengths declared before use but individual records can be randomly accessed by RECORD or BLOCK level I/O requests. All blocks in a contiguous file are physically adjacent, while blocks in a linked file are typically not adjacent (the first word of each block contains the address of the next block). Files can be deleted or created at any time, and are referenced by name. Table 3-1 summarizes the features and benefits of the DOS/BATCH Monitor.

The user communicates with the Monitor in two ways: through batch streams or keyboard instructions called commands, and through programmed instructions called requests.

Batch streams or keyboard commands enable the user to load and run programs; assign I/O devices or files; start or restart programs at specific addresses; modify the contents of memory locations; retrieve system information such as time of day and date; and dump core. Users can utilize programmed requests, which are macros assembled into the user's program through which the user specifies the operation to be performed by the Monitor. Some programmed requests are used to access input/output transfer facilities, and to specify where the data is, where it is going, and what format it is in. In these cases the Monitor will take care of bringing drivers in from disk, performing the data transfer, and notifying the user of the status of the transfer.

Table 3-1
PDP-11 DOS/BATCH Monitor Features and Benefits

Feature	Benefits to User
Files are catalogued in multi-level file directories.	No file naming conflicts among users.
Files are referred to by name.	Files do not have to be remembered by number.
Files can grow serially.	Files can be created or expanded even when their final size is not known.
Files can be as large as the storage device can accept.	No logical limit on the size of files.
File storage is allocated dynamically on any bulk-storage device.	Files can be deleted or created even at run time for maximum storage efficiency.

(continued on next page)

Table 3-1 (Cont.)
PDP-11 DOS/BATCH Monitor Features and Benefits

Feature	Benefits to User
Monitor subroutines can be swapped into core when needed. Routines need not permanently tie up an area of core.	Much more efficient use of core space for user programs. Free core expands and contracts as Monitor subroutines are used. Space can be reclaimed for user programs. The user can determine which Monitor subroutines will be in core, and when.
Monitor subroutines can be made permanently core resident before or during run time.	The user can tailor the Monitor for his particular needs.
The Monitor is divided into logical modules.	The user can easily and efficiently use the logical pieces of the Monitor for his own needs. He can also easily add his own specialized drivers to the system by following a simple set of rules, and still use the rest of the Monitor with these drivers.
All I/O is interrupt-driven.	Such specialized equipment as communications modems and A/D converters which must be interrupt-driven can be run under the Monitor. Several I/O calls can be handled concurrently.
Device independence.	Any device can be specified by the user in his program, and another device can be substituted by him when his program is being run.
Devices are assigned to one or more datasets.	The user may reassign a device which is used for one purpose (dataset) without changing its assignment for all other purposes (datasets).
Two modes available.	Interactive mode and batch mode allow user great ease of program development.

Other requests access Monitor facilities to query system variables such as time of day, date, and system status, and to specify special functions for devices.

Programs supported by DOS/BATCH, and hence accessible through the Monitor, are listed in Table 3-2.

Table 3-2
Principal DOS/BATCH System Programs

Assembler (MACRO-11)
FORTRAN IV Compiler
File Utility Package (PIP)
Debugging Program (ODT-11R)
Linker (LINK)
Librarian (LIBR)
Text Editor (EDIT-11)
File Compare Program (FILCOM)
Verification Program (VERIFY)
Disk Initialization Program (DSKINT)
File Dump Program (FILDMP)
Core Image Library Update and Save (CILUS)
System Loader (SYSLOD)

1.2 MONITOR CORE ORGANIZATION

Core memory is divided into:

- a user area where user programs are located;
- the stack where parameters are stored temporarily during the transfer of control between routines;
- the free core or buffer area which is divided into 16-word blocks assigned by the Monitor for temporary tables, for Monitor routines called in from disk, and for data buffering between devices and user programs;
- the resident Monitor itself which includes all permanently resident routines and tables;
- the interrupt vectors.

Figure 3-1 is a map of core as organized by the Monitor.

The Monitor dynamically acquires and releases core on the basis of system requirements.

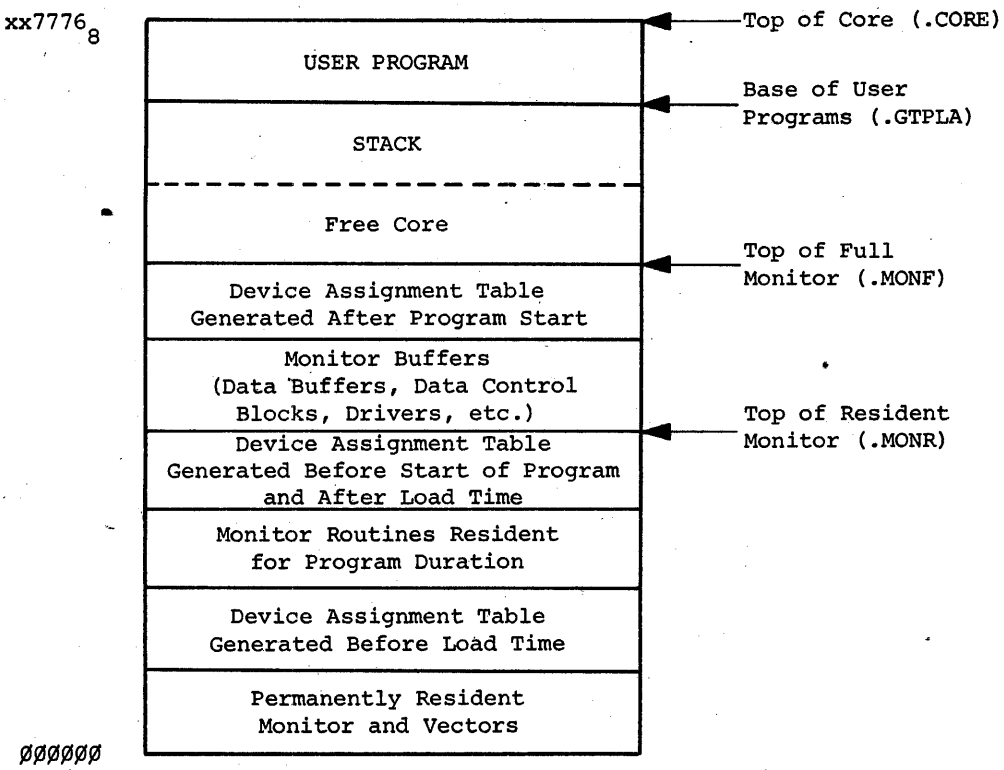


Figure 3-1
The Monitor Core Map

1.3 HARDWARE CONFIGURATIONS

Many minimum hardware configurations for use by the operating system may be derived by choosing one item from each of the five following sets.

- PDP-11 System Building Block with 9ØØ nsec. Core Memory and a Terminal (DECwriter [LA3Ø], Alphanumeric CRT [VTØ5-B], or Teletype[1] [LT33]).
- Cabinets and all Mounting Hardware.
- Bootstrap Loader (BM792-YB or MR-11).
- Choice of Disks (Control Logic Included)

  256K word Fixed Head Disk (RF11/RS11)
  1.2 million word Interchangeable Cartridge Disk (RKØ5/RK11)
  2Ø million word mass storage disk (RPØ3/RP11C)

[1]Teletype is a registered trademark of the Teletype Corporation.

- Choice of Tape Devices (Control Logic Included)

  Dual Drive DECtape (TU56/TC11)
  7- or 9-track Industry Standard Magnetic Tape (TU10/TM11)
  Dual Drive Cassette (TU60/TA11)
  High-Speed Paper Tape Reader/Punch (PC11)

Specific details are available from a sales representative.

1.4 MONITOR MESSAGE

When a message-producing situation (such as a system error) occurs, an error code and an additional word of information are displayed on the terminal. There are five types of message:

1. Action required by the operator
2. Fatal
3. Informational
4. System Program error
5. Warning to the operator

The type of message is identified by being preceded by the letter A, F, I, S, or W respectively. If the system disk should fail and the error message cannot be brought into core, the Monitor halts.

Monitor messages are described in detail in Appendix K.

1.5 STARTING THE MONITOR

The Monitor is called into core from disk by performing the following procedure for systems with the BM792YB:

1. Move HALT/ENABLE switch to HALT position;
2. Load the processor switch register with 173100;
3. Depress LOAD ADDRESS processor switch;
4. Load the switch register with,

   177462 if the system device is RF11 disk,
   177406 if the system device is RK11 disk,
   176716 if the system device is RP11 disk,

5. Move HALT/ENABLE processor switch to ENABLE position;
6. Depress START processor switch.

With the MR11 Bootstrap Loader, the procedure is:

1. Load the processor switch register with:

173ØØ	if the Monitor storage device is RF11 disk,
173Ø11	if the Monitor storage device is RK11 disk,
173154	if the Monitor storage device is RP11 disk,

2. Move HALT/ENABLE switch to HALT position;

3. Move HALT/ENABLE switch to ENABLE position;

4. Depress LOAD ADDRESS processor switch;

5. Depress START processor switch.

The Monitor will load into core and identify itself by printing:

```
DOS/BATCH Vxx-xx
DATE:
```

on the terminal. The user enters the date in the format dd-mmm-yy.

```
 dd = day of the month.
mmm = 3-letter abbreviation of the month.
 yy = year which must be equal to or greater than 71 and less than or
      equal to 99.
```

The Monitor then requests the time of day.

```
TIME:
```

The user specifies the time in the format hh:mm.

```
hh = hour of the day.
mm = minutes.
```

The Monitor is now ready to go into a dialogue mode if the system was initialized to do so, or it is ready to accept an operator command (see Chapter 3-2).

1.6 TERMINOLOGY

The reader should understand the following terms as they apply to DOS/BATCH. An expanded Glossary, with abbreviations, can be found in Appendix I.

A dataset is a logical collection of data which is treated as an entity by a program. Typically, the items in a dataset have a relationship to each other which simultaneously binds them together and distinguishes them from items in other datasets. For example, the records in the Object dataset produced by the assembler are clearly related to each other and are clearly distinct from the

listing dataset produced by the same assembler. A parameter file and a source file, when presented successively to the assembler, might be viewed as a single dataset, however.

Typically, each dataset is associated with exactly one link block (see Section 3-3.9.1), although a link block can be associated (successively, not simultaneously) with more than one dataset. For example, when the assembler finishes processing one dataset and returns for another command, the new input will constitute a new dataset, but the same link block will be used.

Examples of datasets are:

- all or part of a file on a file-structured device;
- one or more paper tapes in a paper tape reader;
- a deck of cards, terminated by an EOF card;
- three lines of keyboard data, a disk file, and a paper tape; which are read in sequence by the assembler and are viewed as the source input dataset.

A device is any PDP-11 peripheral supported by the Monitor.

A device controller can support one or more devices.

A file is a physical collection of data which resides on a directory device (e.g., disk or DECtape) and is referenced by its name. A file occupies one or more blocks on a directory device. See Section 2-2.3.1.2 for more information.

Bulk storage devices which allow data to be stored by name rather than by physical location are called file-structured or directory devices. Devices such as paper tape equipment and terminals which cannot support a file structure are called non-directory devices or non-file-structured devices.

A block is a group of adjacent words of a specified size on a device; it is the smallest system-addressable segment on the device. If the blocks comprising a file are physically adjacent to each other, the file is said to be contiguous; if the blocks of the file are not physically adjacent, the file is said to be linked.

A line is a string of ASCII[1] characters which is terminated by a LINE FEED, FORM FEED or VERTICAL TAB.

---

[1]ASCII represents American Standard Code for Information Interchange.

File structure refers to the manner in which files are organized. Specifically, each of a user's files is given a unique name by the user. Each user on a file-structured device is assigned a User File Directory (UFD) in which each of his files is listed by name and location. Each UFD is then listed in a Master File Directory (MFD) which is unique to a specific device unit.

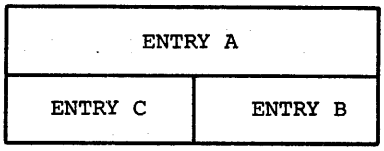
Throughout this manual the terms file structure directory, directoried device, file-structured device and non-file-structured device all refer to the DOS/BATCH Monitor-imposed file structure upon disk devices and/or DECtape. The file structures on magtape and cassette tape are independent of the disk or DECtape file structure. Magtape tape Open (EMT 63_8) and cassette tape Open (EMT 71_8) impose the file structures upon the appropriate devices.

1.7 STANDARDS FOR TABLES

A table is a collection of data stored in sequential memory locations. A typical table as represented in this Part is shown below. This table is two words long, and is referenced by the symbolic address TABL:. The first entry is at location TABL and contains ENTRY A, which might be coded as .WORD AYE in the user's program. The second word of the table, at address TABL+2, is divided into two bytes. The low-order byte (address TABL+2) contains ENTRY B, and the high-order byte (address TABL+3) contains ENTRY C. They might be written into a program as .BYTE BEE,CEE.

a) Representation in manual

TABL:



b) Representation in program listing:

```
TABL:   .WORD AYE       ;ENTRY A
        .BYTE BEE,CEE   ;ENTRY B, ENTRY C
```

Note that the first byte specified is stored at the rightmost available byte.

Unless stated, all numbers in the text and examples are in octal form.

PART 3

CHAPTER 2

MONITOR KEYBOARD COMMANDS

2.1 INTRODUCTION

This chapter shows how the DOS/BATCH Monitor looks to the user as he sits at the terminal (i.e., the Teletype, DECwriter, etc.). The user is communicating with the DOS/BATCH Monitor while running system, utility, and user programs.

For DOS/BATCH in the interactive mode, the primary input and output device is the user's terminal or teleprinter (keyboard and printer). Through the terminal keyboard, the user can communicate with:

- the Monitor,
- a system or utility program (Macro, PIP, Editor, etc.), or
- a user program written to run under DOS/BATCH.

The console terminal is used for user input and system output.

In communicating with the Monitor, the keyboard is used as a control device to allocate system resources, move programs into core, start and stop programs, and exchange information with the system. Data from the keyboard may be transferred to a buffer in the user program or it may be processed immediately by the DOS/BATCH Command String Interpreter (CSI) as explained in Chapter 3-6. In this chapter, the CSI is described only as it applies to the formatting of Monitor keyboard commands.

When the system is ready for input from the keyboard, a single character is printed on the terminal. The following conventions apply:

Character	Meaning
$	The system is idle, waiting for a Monitor command.
.	The Monitor is waiting to continue or abort a task.
#	A system, utility, or user's program requests a command through the CSI.
*	A system program requests direct input, i.e., not through the CSI.

In this chapter, we are concerned only with the $ and . characters. The # and * characters are explained in the individual parts of this handbook.

The $ and . indicate that the Monitor is waiting for a keyboard command from the user. Note, however, that some commands may be issued only to a $ and some only to a . and that each command has different limitations; these are discussed with each command in Section 3-2.8.

2.1.1 Monitor Commands by Function

A number of keyboard commands are provided for communication with the Monitor. These commands are briefly identified by function in Table 3-3 and are fully described in Section 3-2.8.

Table 3-3
Monitor Commands by Function

Function	Command
Establish identity of user	LOGIN
Terminate a session before leaving the system	FINISH
Enter or retrieve date	DATE
Enter or retrieve the time-of-day	TIME
Load and execute a program	RUN
Load a program	GET
Start a program which has been loaded	BEGIN
Resume a program that is waiting for user action	CONTINUE
Assign an I/O device or a file at run-time	ASSIGN
Inspect or modify individual memory locations	MODIFY
Save a program in core for later use	SAVE
Dump memory data on the terminal	DUMP
Suppress or resume echoing of keyboard input	ECHO
Suppress or resume terminal output	PRINT
Start the program just loaded at its ODT entry point	ODT
Stop a program	STOP
Suspend a program	WAIT
Restart a program that has been running	RESTART
Terminate a keyboard or paper tape dataset	END

2.1.2 When Monitor Commands are Legal

Each command performs a specific function, is legal to use under specific conditions, and often alters the state of the system, as shown on the following page.

Command	Legal When	State Induced
ASSIGN	any time	no change
BEGIN	program loaded and stopped	program running
CONTINUE	program loaded and waiting	program running
DATE	any time	no change
DUMP	any time	no change
ECHO	program running	no change
END	program running	no change
FINISH	no program loaded	logged out
GET	no program loaded	program loaded and stopped
KILL	program loaded	program stopped and unloaded
LOGIN	not logged in	logged in
MODIFY	any time	no change
ODT	program loaded	program running under ODT
PRINT	program running	no change
RESTART	program loaded and stopped/waiting	program running
RUN	no program loaded	program loaded and running
SAVE	program loaded and stopped	no change
STOP	program running	program stopped
TIME	any time	no change
WAIT	program running	program waiting

A program is loaded if the user has typed RUN or GET but not KILL, and as long as the program has not executed a .EXIT call (see Chapter 3-3).

A program is running if the user has typed RUN or if it has been loaded and you have typed BEGIN, CONTINUE, RESTART, or ODT.

A program is loaded and stopped if GET but not BEGIN was typed, if it was running and a STOP was typed, or after issuing a fatal error message (see Appendix K).

A program is waiting if it was running and the user typed CTRL/C followed by WAIT, or after the system issues an action error message (see Appendix K).

A program is stopped and unloaded (from core) if the user has typed KILL or if the program issued an .EXIT call (see Chapter 3-3).

2.2 MONITOR MODE AND USER MODE

From the user's point of view, his terminal is in either Monitor mode or user mode. In Monitor mode, each line the user types is sent to the Monitor command interpreter (Transient Monitor). The execution of certain commands (GET or RUN) places the terminal in user mode. The return to Monitor mode is achieved either by a KILL command or an EXIT EMT. When the terminal is in user mode, it becomes simply an input/output (I/O) device for that user. In addition, user programs use the terminal for two purposes: to accept user command strings (user mode) or as a direct I/O device (data mode).

2.3 MONITOR COMMAND INTERPRETATION

When the terminal is in Monitor mode, the user communicates with the system. The system makes several checks before processing commands from the user. For example, if a user who has not logged in types a command that requires him to be logged in, the system responds with the message:

PLEASE LOGIN

meaning that the command was illegal and was not executed. When a command is issued that requires the job to use more core than is available, the system responds with the message:

NO CORE!

and the user's command is not executed.

All Monitor messages are shown in Appendix K.

2.4 USER IDENTIFICATION AND PROTECTION CODES

2.4.1 User Identification Code (UIC)

Each user of the system is normally assigned a User Identification Code (UIC) by the system or installation manager. The UIC is first used when logging into the system, as explained in Section 3-2.7.

The format of the UIC is:

[nnn,nnn]

where nnn represents a pair of 1- to 3-digit octal numbers, each of which may have a value between 11 and 376 (Ø-1Ø are reserved for special use). The value to the left of the comma represents the user-group number, while the value to the right represents the user's number within the group. Thus, if the user is assigned a user number 27 within group 34, he would enter [34,27] for [UIC]. Except when logging in, the UIC is always delimited by the left and right square brackets.[1]

The maximum number of UIC's that a directory device can contain is dependent upon the block size of the device. The maximum capacity is the integer result of subtracting one from the block size and dividing that by four as illustrated on the following page.

---

[1]On Teletype terminals, the left bracket is typed using SHIFT/K; the right bracket is typed as SHIFT/M.

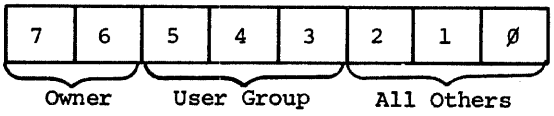
MAXIMUM UIC CAPACITY OF DIRECTORY DEVICES

DIRECTORY DEVICE	BLOCK SIZE (in words)	MAXIMUM NUMBER OF UIC's
RC11	64	15
RF11	64	15
RK11	256	63
RPØ3	512	127
DECtape	256	63

The user identification code is used in connection with file storage and protection. When a UIC does not appear in a command string, the UIC specified in the last LOGIN command is assumed.

2.4.2 Protection Codes

DOS/BATCH provides file security (see Section 2-3.4.1.2) by means of file protection codes. Each file's record in the user's directory (or, on magnetic tape, in the file's header label) includes a binary protection code, in which an octal digit defines the permissable operations for specific classes of users. The four modes of operation are: running, reading, writing, and deleting. The three classes of user are: the owner, the user group, and all other users. The protection code, which is specified as an octal value argument to the PROTECT (/PR) switch, is treated as three fields corresponding to owner, user group, and all others. Each field is assigned an octal digit as illustrated below.



Owner: Bit 6 = 1 indicates that the Owner of the file cannot write on or delete the file. This is a safeguard to prevent inadvertent deletion or overwriting.

Bit 7 (not used)

User Group and All Others:

PROTECTION CODE (octal value)	OPERATION DELETE	WRITE	READ or RUN
Ø	yes	yes	yes
1	no	yes	yes
2 or 3	no	no	yes
4,5,6, or 7	no	no	no

Yes indicates that the operation is allowed; No indicates that the operation is not allowed.

Examples Illustrating Protection Codes:

Protection Code	Description
233	Allows any access by the owner; the user group and all others may read or run but not delete or write on the file. This is the default code provided by the system.
377	Allows the owner to read or run but not delete or write on the file. Neither the user group nor anyone else may have access to the file.
Ø13	Allows the owner to delete, write, read or run. The user group can only write, read or run. Anyone else may read or run but not delete or write on the file.

2.5 FILENAMES AND FILENAME EXTENSIONS

User program files are named with a certain convention, much the same as a person is named. For example, the first name is the filename and the second name is the filename extension. By convention, the filename and extension are separated by a period. For example:

```
FILNAM.EXT
```

could be a legal filename and extension. Note that the filename and extension cannot have embedded blanks (spaces) because a space will be interpreted as a delimiter.

Filenames can consist of from one to six alphanumerics; all characters after the sixth are ignored. However, the first character must be alphabetic. The filename extension can consist of from one to three alphanumerics. The extension is generally used to indicate the type of information in the file. For example:

File	Could be:
MAIN.FTN	a FORTRAN file named MAIN
SAMPLE.MAC	A Macro source file named SAMPLE
TEST1.TMP	a temporary file named TEST1
NAME.OBJ	a relocatable binary file named NAME

The list of standard extensions used by the DOS/BATCH system are shown in Appendix E.

User program files are identified by their filename.extension and the UIC. Thus, different users may use the same filename.extension, and as long as they are created under different UIC's the files would remain distinct and separate.

The asterisk (*) character can be used in a command string to replace either the filename or filename extension specification. The asterisk can be read as "all files" with the filename or filename extension indicated. For example:

*.TMP	indicates all files with the extension .TMP
FILE.*	indicates all files with the name FILE

The asterisk can appear in both positions:

.

which denotes all files on the specified device belonging to the user identification code specified (or the current user UIC if no UIC is specified).

The asterisk feature can generally be used in all transfer operations, all directory listing operations, deletion operations, protection operations and rename operations. See Part 12 for exceptions.

2.6 SPECIAL KEYBOARD CHARACTERS

There are several special keyboard characters recognized by the Monitor command string scanner that cause specific functions to be performed. These keyboard characters are explained below.

2.6.1 The RETURN Key

The RETURN key is used to terminate a keyboard command and to advance the terminal paper one line. Typing the RETURN key produces a carriage return and line feed action on the terminal.

As characters are typed, they are transferred into a buffer where they are stored until the RETURN key or another special keyboard character is typed. When the RETURN key is typed, the data on that line is transferred to and processed by the CSI.

All legal command strings are terminated by the RETURN key.

2.6.2 The RUBOUT Key

The RUBOUT key is used to correct typing errors. Typing the RUBOUT key causes the last character typed to be deleted; successive characters may be deleted by repeated rubouts. The Monitor prints the deleted characters delimited by backslashes. For example, if the user meant to type ASSIGN but typed ASIS instead, the error could be corrected by typing two rubouts and then the correct characters. The printout would be:

```
ASIS\SI\SIGN
```

Notice that the deleted characters are shown in the order in which they are deleted.

2.6.3 The CTRL/C Keys

The CTRL/C key combination is typed by holding down the CTRL key while typing the C key. When CTRL/C is typed, the Monitor is alerted to accept a command from the keyboard. CTRL/C is echoed on the teleprinter as ↑C, carriage return, line feed, and period.

CTRL/C interrupts terminal output or keyboard input in a user program. Monitor action on a CTRL/C is not taken until any current Monitor command is completed because the keyboard interrupt is turned off.

CTRL/C puts the Monitor in listening mode only. If it is desirable to stop the function of the operating program, the STOP command should be used.

If a second CTRL/C is typed before the RETURN key which terminates the command is pressed, the input entered on the current line is erased, a fresh ↑C is printed, and the Monitor awaits a new command. The second CTRL/C merely deletes the line (similar to a CTRL/U); it does not kill the program.

2.6.4 The CTRL/U Keys

The CTRL/U key combination is typed by holding down the CTRL key while typing the U key. The combination CTRL/U may be used for either of the following purposes:

1. To cancel a line of input before it is sent. In this case, CTRL/U is echoed on the terminal as ↑U, carriage return, and line feed.
2. To suppress printing of output at the terminal (except that generated by a batch stream). In this case, CTRL/U is not echoed.

When CTRL/U is typed, the line on which it is typed is deleted; the system responds with a carriage return and line feed so that the line (command) may be typed again.

CTRL/U is echoed on the terminal as ↑U, carriage return, and line feed.

2.6.5 The Semicolon Key

When in Monitor mode (i.e., following a CTRL/C), the semicolon (;) key causes subsequent characters on the line to be treated as a comment. It effectively puts the keyboard off-line so that all characters following the semicolon are printed on the teleprinter but no Monitor action is taken.

2.6.6 The ESCAPE Key

The ESCAPE key (ASCII Ø33 octal) may be used to pass special keyboard characters to a running user program. When the CSI detects the ESC key it passes the next character directly to the user program. The use of this feature is under programmer control.

2.6.7 How Keyboard Characters are Processed

As characters are typed they are stored in the keyboard buffer (about 85 characters capacity) pending termination of the line with a RETURN, CTRL/C, or CTRL/U, which transfers the line of characters to the Monitor buffer.

When a RUBOUT is processed, it remains in the keyboard buffer and the character which it deletes is replaced with another RUBOUT. Since RUBOUTs are not removed until the line is transferred to the user, the capacity of the keyboard buffer may be exceeded if the sum of normal characters plus RUBOUTs is greater than 85. When this occurs, only CTRL/U is accepted; all other characters are discarded and not echoed. This is done to maintain economy of core and to ensure that characters such as CTRL/C and CTRL/U can be processed correctly, even when they appear at the end of a very long line.

2.7 GETTING ON THE SYSTEM

In order to gain access to the system, the user must log in with the LOGIN command (see Section 3-2.8.11). First, ensure that the terminal is connected to the system (see Appendix H). The LOGIN command is issued in response to the Monitor's $. If none exist on the terminal paper, type the RETURN key and a $ will be printed by the Monitor; if not, a new Monitor must be loaded as described in the DOS/BATCH System Manager's Guide.

In response to $, the user should issue the LOGIN command with his User Identification Code (UIC) (see Section 3-2.4). For example:

```
$LOGIN 2ØØ,2ØØ
DATE:-2Ø-OCT-72
TIME:-1Ø:41:16
$
```

In response to the LOGIN command, the Monitor prints the current calendar date and time-of-day followed by the $, indicating that the system is ready for a Monitor command from the user.

Only one user can be logged in at a time. The LOGIN command will be rejected when it is given before the previous user has logged out with the FINISH command.

2.8 MONITOR KEYBOARD COMMANDS

A keyboard command to the Monitor consists of two parts: a command name and possibly one or more command arguments. A command name is a string of two or more letters; all letters after the first two and up to a command name delimiter (space or comma) are optional and are ignored.

Monitor keyboard commands are typed in response to a dollar sign ($) or a period (.), which is printed by the system. Generally speaking, the $ indicates that the Monitor is waiting for a new task, and the . indicates that the Monitor is waiting to continue or abort a previously assumed task.

Although the commands are arranged in alphabetical order for ease of reference, they can be divided into functional groups for ease of learning. These groups with their associated commands are as follows:

- Command to allocate system resources:

  ASSIGN

- Commands to manipulate core images:

  RUN GET
  DUMP SAVE

- Commands to start a program:

  BEGIN CONTINUE
  RESTART

- Commands to stop a program:

  STOP WAIT
  KILL

- Commands to exchange information with the system:

  DATE TIME
  LOGIN MODIFY
  FINISH

- Miscellaneous commands:

  ECHO PRINT
  END ODT

The following conventions apply to all Monitor commands:

1. All commands are terminated with the RETURN key.
2. The command name is separated from its argument (dataset specifier, etc.) by one or more spaces.
3. All characters in a command are interpreted by the CSI; thus, no embedded blanks are allowed.
4. The UIC is always enclosed within square brackets, [ ], except when used with the LOGIN command.

The proper format for each command is given in the discussion of each command in this section. The following conventions apply to the command formats shown in this section.

The dataset specifier may be represented by the expression:

[dev:] filnam[.ext][[uic]]

where

dev: is a legal device mnemonic and colon (see Appendix C).

filnam is a filename of up to six alphanumerics.

.ext is a period and filename extension of up to three alphanumerics.

uic is the user's identification code in the form:

[group number, user number]

The brackets are part of the UIC and keyed as part of it.

The logical name is the name given by the user to the dataset in Link Block word LNKBLK+2 (see Chapter 3-3).

If for any reason a command cannot be executed satisfactorily, an appropriate message will be printed on the terminal and the command will be ignored. These messages are shown in Appendix K.

ASSIGN 2.8.1 The ASSIGN Command

Format:

AS[SIGN]Δ[dataset specifier, logical name]

Purpose:

This command assigns a physical device (and a filename when the device is file-structured) to the dataset identified by "logical name". If a dataset specifier is included in the ASSIGN command, a logical name must be specified also.

Any filename specified for a non-file-structured device is ignored.

Note that a device is assigned to a dataset, and that reassigning it for one dataset does not reassign it for all datasets.

The ASSIGN command overrides any assignment made in the program's internal control blocks (Link and Filename Blocks). The ASSIGN command is not needed if the program makes its own provisions for obtaining this information; e.g., by specifying defaults in its control blocks or by requesting a command string, as is done with the # symbol in the DOS/BATCH system programs.

An ASSIGN with no argument cancels all ASSIGNments previously made by the current user, i.e., since the last LOGIN command.

The ASSIGN command can be given at any time the Monitor is in core. Consider the following:

1. If ASSIGN is given before a program is loaded, the device assignment will remain in effect until another ASSIGN is given with the same logical name or with no arguments, or until the Monitor itself is reloaded (as with a FINISH command or hardware reboot). ASSIGN, given at this time, enables the user to specify an assignment which will apply to several programs.

2. If ASSIGN is given after a program is loaded and before it has started running (i.e., after a GET command), the assignment will remain in effect as long as the program is in core, or until another ASSIGNment is performed. When the program disappears (by an .EXIT request or a KILL command), the assignment is released.

3. ASSIGN may also be given after a program is running. For example, as a recovery from an

   AØØ3 (illegal or nonexistent device code)

   message, the user would do an ASSIGN followed by a CONTINUE. The assignment will remain in effect as long as the program is in core, or until the programmer reassigns the dataset, or until he restarts the program with a BEGIN command.

   Doing an ASSIGN in this manner is provided for such emergency situations, but is not recommended as standard practice because it causes an extra buffer to be allocated from free core, and it will be effective only if the program has not already INITed the dataset to some other device.

Examples:

```
AS DTØ:,3
ASSIGN DKØ:INPUT.DAT[1,5],INFIL
AS SY:OUTPUT.DAT[14,123],OUTFIL
ASSIGN
```

BEGIN 2.8.2 The BEGIN Command

Format:

```
BE[GIN]Δ[address]
```

Purpose:

The BEGIN command starts the execution of an already loaded program at the stated address. If no address is specified, the normal start address will be used. This command is valid only if a program is already in core.

BEGIN is used after a GET, a STOP, or following a fatal error condition. It removes all core allocations of buffers, device drivers, and assignments made dynamically, and the stack is cleared before control is passed back to the program. If any files are under creation at this time, they are deleted.

To start a program at its normal start address, type:

```
BE
```

To start a program at absolute address 3446, type:

```
BEΔ3446
```

After a Program Crash:

The BEGIN Command is provided not only as a means of starting a program loaded by GET but also to enable the user to try again after a program crash, hopefully with a clean slate. At the time of the crash, the program may already have opened but not closed output files and the subsequent request to reopen after a restart could then lead to other failures because these files now exist. To prevent this, the BEGIN processor tries to delete the files, but not by the normal Monitor process since this could mean writing out bit-maps which are currently in core and must be suspect because of the crash. Instead, it merely removes the names of the files from the appropriate device directory, and if these are on disk, unlinks any blocks so far allocated; for safety it does not touch the bit-maps already stored on the device. In almost all cases, this procedure suffices. However, the following implications should be noted.

1. This automatic deletion by BEGIN will not suit a user who has already amassed considerable data in one of his output files and cannot replace it if he starts over. In this case, KILLing the program to save his data under a different filename might be a more appropriate action. However, the user should then realize that further errors may occur from the use of this data.

2. It is possible that by the time of the crash the program may have produced a fairly long file. On a DECtape for which there is only one bit-map, this is no problem. A disk, however, requires several bit-maps and the allocation of some of the blocks for the file may already be permanently recorded because the appropriate bit-map has been filled and has been replaced in core by another. Since BEGIN does not change the maps, these blocks will not be released for further use. Facilities for recovering this space are provided by the system program VERIFY. A series of situations such as this can, after a time, result in the disk becoming full even though the known files are not seen to occupy the whole capacity. The user should in this case consider whether or not he should chance disk-corruption and use KILL rather than BEGIN. The user can then delete the file by using PIP to avoid the build-up of the nonavailable blocks described.

3. Some programs cannot be restarted with BEGIN (i.e., after having been started, they cannot be restarted with BEGIN). A FORTRAN program is an example. In general, a program must be self-initialized if BEGIN is to be used in this way. Also, since the Monitor will try to clean up core and delete files, reBEGINing a program which was badly out of control may lead to undesirable results. Thus, use BEGIN only if there is no other alternative.

2.8.3 The CONTINUE Command — CONTINUE

Format:

CO[NTINUE]

Purpose:

This command is used after a WAIT command or a recoverable error condition (operator action message) to resume program operation at the point where it was interrupted.

CONTINUE is valid only if a program is already in core.

2.8.4 The DATE Command — DATE

Format:

DA[TE]Δ[date]

Purpose:

The DATE command may be used to obtain the current calendar data and to enter a date value from the keyboard; the data is printed in the dd-mmm-yy format.

To obtain the current calendar date, simply type the DATE command followed by the RETURN key. For example:

```
$DATE
28-FEB-74
$
```

To enter a date value from the keyboard, type the DATE command, the desired date value, and then the RETURN key. For example:

```
$DATEΔdd-mmm-yy
```

putting the desired date value in place of dd-mmm-yy. The entered date value is returned in response to subsequent DATE commands until another date is given. Any invalid date is rejected.

DATE is valid at any time.

DUMP 2.8.5 The DUMP Command

Format:

DU[MP]{Δ ,}LP:[,O][,{start addr Ø}[,end addr]]

Purpose:

The DUMP command is used to print on the line printer an absolute copy of the contents of the specified core area, formatted in octal. The core image is not altered.

The argument O specifies the dump to be output from core. An O is assumed on default, but the comma is required.

The argument Ø is assumed if no "start address" is specified and the highest word in core is assumed if no "end address" is specified.

DUMP is valid at any time. If given while a program is running, the operation of the program will be suspended for the time required to effect the dump.

The syntax of the DUMP command was chosen to facilitate later expansion and flexibility of the command.

2.8.6 The ECHO Command — ECHO

Format:

EC[HO]

Purpose:

The ECHO command may be used to suppress and restore keyboard echo, i.e., characters typed by the user will not appear on the terminal printer. A subsequent ECHO command turns the echo feature on again. The terminal as an output device for the program or the Monitor is not affected by this command.

ECHO is valid only when a program is running in core and using the keyboard as an input device.

2.8.7 The END Command — END

Format:

EN[D]Δ{KB / PT}

Purpose:

The END command is used to terminate the use of the keyboard or low-speed paper tape reader as an input device. The command tells the Monitor "there is no more input from the device". The command effectively generates an end-of-file (EOF) from the keyboard.

When no device is specified in the command, KB is assumed.

The following actions are required with this command

1. Type CTRL/C to obtain the Monitor's attention. Since the console is being used for program input (data mode), the Monitor is not expecting a command.
2. Issue the END command (with appropriate argument).
3. Type the RETURN key twice. Two RETURNs are required to return to the Monitor.

For example: (where ↑C = CTRL/C, and (CR) = RETURN)

```
↑C
.END KB     (CR)     (CR)
```

END is valid only when the specified device is being used as an input device.

FINISH 2.8.8 The FINISH Command

Format:

FI[NISH]

Purpose:

The FINISH command informs the Monitor that the current user is leaving the system and a new copy of the resident Monitor is "booted" into core.

FINISH is valid only when no user program is in core. Therefore, unless the last character on the teleprinter is a $, the user should precede a FINISH with CTRL/C followed by KILL. For example, the printout might be:

```
↑C
.KILL
$FINISH
TIME:-16:42:ØØ
DOS/BATCH VØ9-xx
$
```

In response to a FINISH, the Monitor prints the time and then the newly booted Monitor identifies itself. The system is now ready for a user to log in.

GET 2.8.9 The GET Command

Format:

GE[T]Δdataset specifier

Purpose:

The GET command loads the specified file from the specified device. When a device is not specified, the system device is assumed.

GET is valid only when no program is in core.

The user should use a BEGIN or ODT command to commence execution.

KILL 2.8.10 The KILL Command

Format:

KI[LL]

Purpose:

The KILL command stops the execution of the current program after closing all open files and completing any unfinished I/O. It then returns control to the Monitor. A RESET instruction is performed during the processing of KILL, thus initializing all I/O.

KILL is valid only when a program is in core.

To resume operations, the user must reload the program or load another with RUN or GET.

2.8.11 The LOGIN Command — LOGIN

Format:

LO[GIN]Δuic

Purpose:

The LOGIN command enables a user to gain access to the system. LOGIN requires a UIC as its argument (see Section 3-2.4). The UIC indicates which of the directories on each file-structured device will be directly available to the user.

Here the UIC is not enclosed within the square brackets; its format is simply

nnn,nnn

specifying group number and user number respectively.

LOGIN is valid only when there is no program loaded in core and provided no user is logged in.

2.8.12 The MODIFY Command — MODIFY

Format:

MO[DIFY]Δoctal address
octal address/contents: [new contents]

Purpose:

This command allows the user to display and make changes to the contents of the absolute memory location specified by "octal address" in the command line. When the RETURN key is typed at the end of the command line, the system responds by

printing the contents of that address. At this point, the user can type one of the following ((CR) = RETURN key; (LF) = LINE FEED key):

(CR)	leaves the contents unmodified.
new contents (CR)	changes contents to new contents.
(LF)	takes similar action as CR and then prints the contents of the next memory location.
new contents (LF)	changes the contents to the new contents and prints the contents of the next memory location.

For example, to change the contents of location 4ØØØØ:

```
$MODIFYΔ4ØØØØ     (CR)
4ØØØØ/Ø164Ø6:  1Ø4Ø6     (CR)
```

Then to examine the contents of 4ØØØØ:

```
$MOΔ4ØØØØ     (CR)
4ØØØØ/Ø1Ø4Ø6:     (CR)
```

To examine the contents of locations 4ØØØØ and 4ØØØ2, the sequence would be:

```
$MOΔ4ØØØØ   (CR)
4ØØØØ/1Ø4Ø6Ø:     (LF)
4ØØØ2/ØØØØØ3:
```

Entry of an address outside the available core memory as part of the original MODIFY command will cause an error, and the command will be rejected.

MODIFY is valid at any time.

ODT 2.8.13 The ODT Command

Format:

OD[T]Δ[{R / K}]

Purpose:

The ODT command starts the execution of the ODT-11R Debugging Program. The argument specifies which ODT start address is to be used:

Argument	Starts at	Action
(none)	START+Ø	Clears ODT breakpoint table without resetting breakpoints.
R	START+2	Clears ODT breakpoint table after replacing old instructions at breakpoints.
K	START+4	Leaves breakpoints exactly as they are.

This command begins execution at the ODT entry point of the user's load module. The user must have linked ODT-11R with his program and must have identified his program to the Linker with the /OD switch.

To reset all breakpoint locations at their former instructions and restart ODT, the user would type:

```
$ODΔR
```

ODT is valid only when ODT-11R is linked to a program and both are in core. The program may or may not be running.

2.8.14 The PRINT Command **PRINT**

Format:

```
PR[INT]
```

Purpose:

The PRINT command may be used to suppress and restore terminal printing when the terminal is used as an output device by a user program. Each PRINT command cancels the effect of the previous PRINT command.

PRINT is valid only when a program is running in core and is using the terminal as an output device.

2.8.15 The RESTART Command **RESTART**

Format:

```
RE[START]Δ[address]
```

Purpose:

The RESTART command permits a program to be restarted. As shown, the user may optionally supply an address at which the program is to be restarted. If no

address is specified, the address set by the .RSTART programmed request is assumed if a .RSTRT request has been issued by the program (see Section 3-3.6.32).

If neither address is specified, the command is rejected.

RESTART is valid only when a program is already in core.

Before the program is restarted, the stack is cleared, any current I/O is stopped, and all internal busy states are removed. Buffers and device drivers set up for I/O operations will, however, remain linked to the program for future use.

RUN 2.8.16 The RUN Command

Format:

RU[N]Δdataset specifier

Purpose:

The RUN command loads into core the specified program from the specified device and starts its execution at the normal start address. RUN is equivalent to a GET command followed by a BEGIN command.

When no device is specified in the dataset specifier, the system device (disk) is assumed.

The sequence in which the Monitor performs its search for the specified program depends on the existence and type of filename extension and on the UIC. Various forms of the RUN command are shown below with the search sequence performed by the Monitor.

- RUNΔFILE

  ```
  Attempt 1 -- FILE.LDA [current uic]
  Attempt 2 -- FILE.LDA [1,1]
  Attempt 3 -- FILE [current uic]
  Attempt 4 -- FILE [1,1]
  ```

- RUNΔFILE.EXT

  ```
  Attempt 1 -- FILE.EXT [current uic]
  Attempt 2 -- FILE.EXT [1,1]
  ```

- RUNΔFILE[nnn,nnn]

  ```
  Attempt 1 -- FILE.LDA [nnn,nnn]
  Attempt 2 -- FILE [nnn,nnn]
  ```

- RUNΔFILE.EXT[nnn,nnn]

  ```
  Attempt 1 -- FILE.EXT [nnn,nnn]
  ```

If all attempts fail to find the file, a NO FILE message is printed at the terminal.

Searching for the LDA extension first exploits the fact that both the Linker and the SAVE command produce LDA extensions, unless the user specifies otherwise.

RUN is valid only when there is no program in core.

2.8.17 The R System Program Command — R SYSTEM

Format:

R Δ [dev:]filename

Purpose:

The R System Program command loads into core the specified system program from device SY: and starts execution at the normal start address. R is equivalent to the RU command with the dataset specification for the program in UIC [1,1] and an extension of .LDA. Attempts to specify either UIC or extension causes the command to be rejected.

The advantage of R is that it saves the time required to search for the program in the user's program directory. The search starts in the system program directory.

2.8.18 The SAVE Command — SAVE

Format:

SA[VE]Δ[dataset specifier][/RA:low:high]

Purpose:

The SAVE command writes the program in core onto the device in loader format. The core image is not altered. SAVE is valid only when a program is in core but not running, i.e., immediately after loading with a GET command or after being halted by either a STOP command or a fatal error.

If no dataset specifier is given, the SAVE processor will automatically set up a file called SAVE.LDA on the system disk after it has deleted any current file of the same name. If the user wishes to retain the current file, he must first rename it using PIP. If the dataset specifier is given, the file named must not already exist or the command will be rejected. The system disk is assumed by default if the dataset specifier contains only a filename. When the filename is specified, the extension should also be specified.

Normally it is expected that the user will only wish to save his program area. If this is the case, the range need not be given and the new file will begin from the program's low limit and extend to the top of core. If any other area is to be saved, the user should include the following at the end of the command:

```
/RA:low:high
```

where /RA is the range switch, and low and high define the limits required (each being valid octal word-bound addresses). The saved image will be preceded by the same communication information as that for the original program loaded, except that any information about the resident EMT modules will be lost.

The SAVE processor will endeavor to get an extra 256-word buffer in order to satisfy the command. If this request cannot be granted because of insufficient free core, the command will be rejected. The user is therefore advised to use this facility only after he has released any datasets currently established.

Once the SAVE command has been syntactically verified, any errors will be handled by the SAVE processor, which will print a relevant message and return to Monitor mode:

DEVICE FULL	End of output medium reached
FILE ERROR xxx	File structures error as indicated by xxx = file status byte

NOTE

Overlayed programs cannot be saved.

STOP 2.8.19 The STOP Command

Format:

```
ST[OP]
```

Purpose:

This is an emergency command to stop the program and to abort any I/O in progress by doing a hardware reset. The program may be resumed with either the BEGIN or RESTART command.

STOP is valid only if a program is in core.

STOP differs from KILL in that KILL terminates the program in an orderly manner and returns control to the Monitor.

2.8.20 The TIME Command — TIME

Format:

TI[ME]Δ[time]

Purpose:

The TIME command may be used to obtain the current time-of-day and to enter a time value from the keyboard. The time is printed in the following format:

hh:mm:ss

meaning hours:minutes:seconds.

To obtain the current time-of-day, simply type the TIME command followed by the RETURN key. For example:

```
$TIME
1Ø:43:27
$
```

The current time-of-day is entered by the system or installation manager, and need not be reentered except when loading a new DOS/BATCH Monitor.

To enter a time value from the keyboard, type the TIME command, the desired time value, and then the RETURN key. For example:

$TIMEΔhh:mm:ss

putting the desired time value in place of hh:mm:ss. The entered time value is returned in response to subsequent TIME commands until another time value is given.

TIME is valid at any time.

2.8.21 The WAIT Command — WAIT

Format:

WA[IT]

Purpose:

The WAIT command suspends the current program and allows any I/O in progress to finish. The program may be resumed with either the CONTINUE or RESTART command.

WAIT is valid only if a program is in core.

PART 3

CHAPTER 3

PROGRAMMED REQUESTS

3.1 INTRODUCTION

The Monitor provides a number of services available to any user or system program. The most prominent of these are input/output (I/O) services. Other services include directory management, retrieval and modification of system parameters, various conversion routines, and a command string interpreter. The I/O services provide for linkage to device drivers, access to files in the file structure, and transfer of data to or from each device.

The user program calls for the services of the Monitor through programmed requests. Programmed requests are macro calls (or the assembly language expansion of such a call) which are assembled into the user program and interpreted by the Monitor at execution time. A programmed request consists of a macro call followed, when appropriate, by one or more arguments. For example:

```
.WAIT #LNKBLK
```

is a programmed request called .WAIT followed by an argument #LNKBLK. The macro request is expanded at assembly time by the MACRO Assembler into a sequence of instructions that passes the arguments to the appropriate Monitor service routine to carry out the specified function, then calls the appropriate Monitor service routine (via an EMT instruction). The assembly language expansion for .WAIT #LNKBLK is:

```
MOV #LNKBLK,-(SP)
EMT 1
```

To use the macro call, it is necessary to tell the assembler that the system definition for the macro is needed. This is accomplished via the .MCALL assembler directive (see Part 6), e.g.,

```
.MCALL     .WAIT
```

which must appear in the source prior to the first use of .WAIT. When .MCALL is encountered, the MACRO Assembler will get the definition of .WAIT from the system macro file (SYSMAC.SML) which is searched first in the current user's disk area, then under user identification code [1,1].

The system macros accept most addressing modes as arguments. They will detect and announce potentially troublesome or unlikely modes to protect the user.

All legal addressing modes will appear without alteration in the expansion. Since the Monitor expects the address of the Link Block on top of the stack at .WAIT time, any of the following macro calls might be appropriate:

```
.WAIT #LNKBLK        ;ADDRESS OF LNKBLK
.WAIT RØ             ;IS IN REGISTER Ø
                     ;ADDRESS OF LNKBLK IS
.WAIT POINTR         ;IN MEMORY LOCATION POINTR
```

The programmed request arguments are parameters or addresses of tables which contain the parameters of the request. These tables are part of the user program, and are described in detail in Figures 3-7 to 3-18.

3.2 TYPES OF PROGRAMMED REQUESTS

Services which the Monitor makes available to the user through programmed requests can be classified into three groups:

1. Requests for input/output and related services,
2. Requests for directory management services, and
3. Requests for miscellaneous services.

Table 3-4 summarizes the programmed requests available under the Monitor. Detailed descriptions of each request can be found in the sections cited in Table 3-4.

Table 3-4
Summary of Programmed Requests

Mnemonic	Purpose	Section
Requests for Input/Output and Related Services:		
.BLOCK	Transfers one physical block of a file between a device and a Monitor buffer.	3.6.5
.CLOSE	Closes a dataset.	3.6.6
.INIT	Associates a dataset with a device driver and sets up the initial linkage.	3.6.21
.OPEN	Opens a dataset.	3.6.25
.READ	Transfers data from a device to a user's line buffer.	3.6.28
.RECRD	Transfers one logical record of a file between a device and a user buffer.	3.6.29
.RLSE	Removes the linkage between a device driver and a dataset, and releases the driver.	3.6.31
.SPEC	Performs special device functions.	3.6.34

(continued on next page)

Table 3-4 (cont.)
Summary of Programmed Requests

Mnemonic	Purpose	Section
.STAT	Obtains device characteristics.	3.6.35
.TRAN	Transfers data between a device and a user buffer, independent of any file structure.	3.6.41
.WAIT	Waits for completion of any action on a dataset.	3.6.43
.WAITR	Checks for completion of any action on a dataset, and provides a transfer address for a busy return.	3.6.44
.WRITE	Transfers data from a user's line buffer to a device.	3.6.45
Requests for Directory Management Services:		
.ALLOC	Allocates a contiguous file.	3.6.1
.APPND	Appends one linked file to another.	3.6.2
.DELET	Deletes a file.	3.6.11
.LOOK	Searches the directory for a particular filename and returns information about the file.	3.6.22
.RENAM	Renames a file. Changes a protection code.	3.6.30
Requests for Miscellaneous Services:		
.BIN2D	Converts one binary word into five decimal ASCII characters.	3.6.3
.BIN2O	Converts one binary word into six octal ASCII characters.	3.6.4
.CORE	Obtains address of highest word in core memory.	3.6.7
.CSI1	Condenses a command string and checks for proper syntax.	3.6.8.1
.CSI2	Interprets one command string dataset specification.	3.6.8.2
.CVTDT	Converts internal date or time to ASCII.	3.6.9
.DATE	Obtains the date.	3.6.10
.DUMP	Dumps contents of memory for specified locations.	3.6.46
.D2BIN	Converts five decimal ASCII characters into one binary word.	3.6.12
.EXIT	Returns control to the Monitor.	3.6.13
.FLUSH	Bypasses lines in the batch stream.	3.6.47
.GTCIL	Gets the base disk address of the CIL.	3.6.14
.GTCLK	Obtains system clock information.	3.6.15
.GTOVF	Obtains and sets the overlay flag.	3.6.16

(continued on next page)

Table 3-4 (cont.)
Summary of Programmed Requests

Mnemonic	Purpose	Section
.GTPLA	Gets the current program load address.	3.6.17
.GTRDV	Obtains RUN device information.	3.6.18
.GTSTK	Gets the current stack base address.	3.6.19
.GTUIC	Gets current UIC.	3.6.20
.MONF	Obtains address of first word above the Monitor's highest allocated free core buffer.	3.6.23
.MONR	Obtains address of first word above the resident Monitor.	3.6.24
.O2BIN	Converts six octal ASCII characters into one binary word.	3.6.26
.RADPK	Packs three ASCII characters into one Radix-5Ø word.	3.6.27.1
.RADUP	Unpacks one Radix-5Ø word into three ASCII characters.	3.6.27.2
.RSTRT	Sets the address used by the RESTART command.	3.6.32
.RUN	Loads programs and overlays.	3.6.33
.STFPU	Sets the floating point exception vector.	3.6.36
.STPLA	Sets the program low address.	3.6.37
.STSTK	Sets the current stack base address.	3.6.38
.SYSDV	Gets Radix-5Ø name of the system device.	3.6.39
.TIME	Obtains the time of day.	3.6.40
.TRAP	Sets interrupt vector for the TRAP instruction.	3.6.42

3.2.1 Requests for Input/Output and Related Services

All user I/O is handled by programmed requests, which provide three different levels of transfer:

- READ or WRITE
- RECORD or BLOCK
- TRAN

The term, request level, refers to the division of work and responsibility between the Monitor and the user. READ/WRITE is the highest level, RECORD/BLOCK is the intermediate level, and TRAN is the lowest level. At the READ/WRITE level, the Monitor assumes most of the responsibility and provides the user with many services. At the RECORD/BLOCK level, the work is more evenly shared, and at the TRAN level, the user assumes most of the responsibility and does most of the work.

Disk I/O handling provides a good example of the function of the three levels. At the READ/WRITE level, the user specifies a block size that is appropriate for the input or output device. The Monitor accumulates read or write requests until the correct block size is reached; then, the physical transfer of data is made. The READ/WRITE level provides extensive formatting, legality checking, and error reporting facilities. Although a small amount of flexibility is lost, the user gains assurance that data on the disk is not going to be harmed.

At the TRAN level, the user can specify any disk location, core location, and the number of words to include in the data transfer. The Monitor does not check the legality of the instruction, except for an invalid address, and does no formatting. If the user issues an incorrect instruction, he may lose data on the disk or corrupt the file structure.

The RECORD/BLOCK requests are at an intermediate level and provide some of the advantages of the lower and higher levels.

Each level uses a sequence of requests to complete the transfer. Note the distinction between READ/WRITE, RECORD/BLOCK, and TRAN as names of transfer levels, and .READ, .WRITE, .RECRD, .BLOCK, and .TRAN as specific programmed requests within these levels.

I/O related services perform special device functions (such as rewinding a magtape) and obtain device characteristics from device status words.

Each request related to I/O services is described in Section 3-3.6.

3.2.1.1 READ or WRITE Level Requests

Most input and output is done at this level. Processing is sequential, in that each read or write is applied to the next record in the file. Records may be in either ASCII or binary mode, and a number of formats are handled by the Monitor. Records may also be of variable length: ASCII records usually contain line terminators while formatted binary records contain byte counts.

READ or WRITE I/O under the Monitor consists of transferring the contents of a dataset between a device and a line buffer via a buffer in the Monitor (see Figure 3-2a). A line buffer is an area set up by the user in his program, into which the user (or the Monitor) places data for output (or input). The line buffer is usually preceded by the line buffer header, in which the user specifies the size and location of the line buffer and the mode (format) of the data.

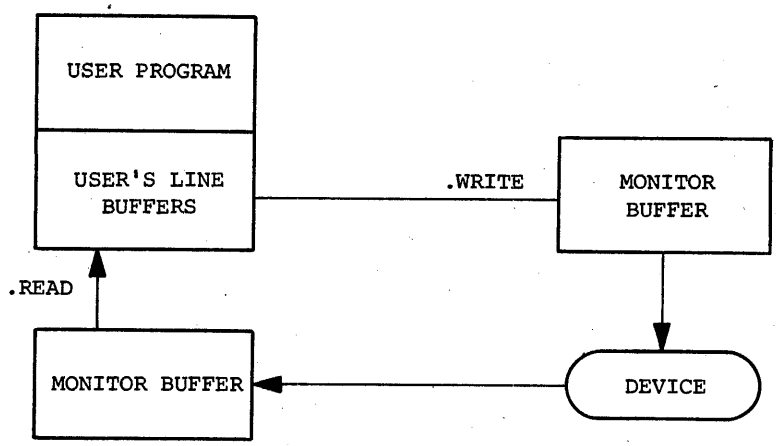


Figure 3-2a
The Transfer Path

```
        .INIT     #LNKBLK

        MOV       #FILBLK,RØ
                                        ;FOR OUTPUT, REPLACE .OPENI
        .OPENI    #LNKBLK,RØ            ; WITH .OPENO
                                        ;ADDRESS OF FILBLK IS IN RØ

        .READ     #LNKBLK,#BUFHDR       ;FOR OUTPUT, REPLACE .READ
                                        ; WITH .WRITE
        .WAIT     #LNKBLK               ;COULD BE REPLACED BY .WAITR
                .
                .
        (Process Data)

        No      END
              OF DATA
                ?
                Yes

        .CLOSE    #LNKBLK
        .RLSE     #LNKBLK

LNKBLK:             (entries)
FILBLK:             (entries)       Tables in User's Program
BUFHDR:             (entries)
```

Figure 3-2b
Sequence of Requests for READ/WRITE

Figure 3-2
.READ/.WRITE Input/Output Transfers

When using READ or WRITE one can specify nine different types of transfer, in two modes: ASCII and Binary. Details are presented in Section 3-4.3.1 and Figure 3-10.

ASCII Modes:	Formatted ASCII Parity - Special
	Formatted ASCII Parity - Normal
	Formatted ASCII Nonparity - Special
	Formatted ASCII Nonparity - Normal
	Unformatted ASCII Parity - Normal
	Unformatted ASCII Nonparity - Normal
Binary modes:	Formatted Binary - Special
	Formatted Binary - Normal
	Unformatted Binary - Normal

To implement a READ or WRITE transfer, the programmer follows the sequence of requests shown in Figure 3-2b. First, the programmer associates the device with the dataset via the .INIT request. The argument of this request is the address of a table called the Link Block. Entries in this table specify the device involved in the approaching transfer so that the Monitor may eventually establish a link between that device and the dataset. The Link Block is described in detail in Figure 3-7. The .INIT request loads the appropriate device driver into the Monitor's free core area, if it is not already there.

Following the .INIT request, the programmer opens a dataset with an .OPENx request. This need be done only if the device being used is a file-structured device. However, it is advisable to use an .OPENx even for a non-file-structured device to preserve the device independence of the program, since it may be desirable to assign the transfer to a file-structured device later. The arguments of this request are the address of the Link Block and a register into which the user has moved the address of a table called the Filename Block (Figure 3-8). Entries in this table describe the file involved in the transfer.

A dataset can be opened for input, for output, for update, or for extension. The last letter of the .OPENx request specifies which type of open is desired.

A .READ (for input) or a .WRITE (for output) follows the .OPENx. Either request causes a transfer to take place between the line buffer and the device via a buffer allocated by the Monitor in its free core area. The arguments of either request are the address of the Link Block for the dataset and the address of the Line Buffer Header (Figure 3-9). The Line Buffer Header specifies the area in the user's core area to or from which the data is to be transferred. During the transfer, the Monitor formats the data according to the transfer mode and formatting characters in the data itself. In most modes, terminating characters indicate the end of a line.

.READ or .WRITE is followed by .WAIT (or .WAITR), which tests for the completion of the last transfer, and passes control to the next instruction when the transfer is complete. Typically, what follows a .WAIT on an input is a subroutine to process the portion of data just read. When the process has been completed, the program checks to see if there is more data; if there is, the program transfers control back to the .READ request and the process is repeated. If all data has been transferred, the .CLOSE request follows to complete any pending action, update any directories affected, and release to free core any buffer space the Monitor has allocated from free core for this dataset. Finally, action on the dataset is formally terminated with the .RLSE request, which dissociates the device from the dataset, and releases the driver. Releasing the driver frees the core it uses provided there is no other claim to the driver from another dataset.

3.2.1.2 RECORD Level Requests

The Record Level request is used for random access to the records in a file. A program which uses Read or Write Level requests can only read or write the next record in the dataset being processed. When Record Level requests are used, the program always has access to any record in the file. See Figure 3-3.

Record Level requests may be used only with contiguous files that reside on file-structured devices. Each of the records in the file must contain the same number of bytes. No formatting is done and no line terminating characters are needed. The length of a record is independent of the block size of the device.

Some consideration must be given to the manner in which a Record Level file is created. Perhaps the most common way to create such a file is by doing an .OPENC (after the file has been allocated) and using the .WRITE request to enter data. Unformatted ASCII and unformatted binary are the suggested transfer modes, since they do not require terminators and do not perform formatting. When such a file is .CLOSEd, a logical end-of-file is established following the last record written. Subsequent processing of the file by .READ or .RECRD will be confined to the area just written. At some later time, the file may be opened for extension (.OPENE) and more data can be written (.WRITE), provided the original space allocated to the file is sufficient to contain it. A second way to create a Record Level file is to start with .OPENU (again the file must have been allocated previously) and to use .RECRD to do the writing. In this mode, the logical end-of-file corresponds to the end of the allocated area.

Before issuing Record Level requests, the program must issue an .INIT request to associate the dataset with a file-structured device. The program is then required to open the dataset. The dataset may be opened in two ways:

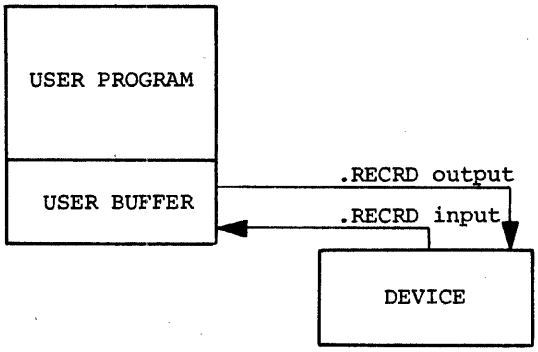


Figure 3-3a
The Transfer Path

```
                    .INIT     #LNKBLK              ;INIT THE DATASET
                    MOV       #FILBLK,R1
                    .OPENU    #LNKBLK,R1           ;OPEN THE FILE
Request next        .RECRD    #LNKBLK,#RECBLK      ;INPUT THE RECORD
record for          .WAIT     #LNKBLK
update if                     |
more                (Process Data)
                              |
                    .RECRD    #LNKBLK,#RECBLK      ;OUTPUT THE RECORD
                    .WAIT     #LNKBLK
                              |
                              v
                         +---------+
                    Yes  |  MORE   |
                    <----|    ?    |
                         +---------+
                              |
                              | No
                              v
                    .CLOSE    #LNKBLK              ;CLOSE THE FILE
                    .RLSE     #LNKBLK              ;RELEASE THE DATASET

LNKBLK:
FILBLK:   }   Tables in User Program
RECBLK:
```

Figure 3-3b
Sequence of Requests for .RECRD

Figure 3-3
.RECRD Input/Output Transfers

- OPENU - This mode is used if the program will write in the dataset. Reading is also permitted. In fact, quite often the program will read a record, update it, and write it back.

- OPENI - This mode is used if no writing will be done. Only reading will be permitted.

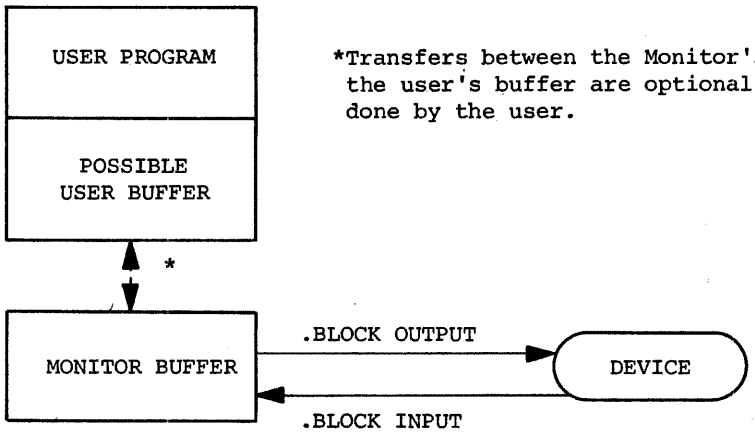
The dataset may then be processed using .RECRD requests. If updating is being done, there will generally be two such requests in each cycle. Otherwise, there will be only one. Each .RECRD request should be followed by a .WAIT (or .WAITR) request. When processing is completed, a .CLOSE request should be issued to ensure that the last record is actually written to the device (for output) and that the directory is updated (if necessary). A .RLSE request is also required, so that the driver can be removed from core if it is not still in use by another dataset. The .RECRD request has a Link Block and a Record Block as arguments. The Record Block specifies function (input/output), buffer address, record length, and record number (see Figure 3-12).

3.2.1.3 BLOCK Level Requests

The Block Level request is used for random access to the physical blocks in a file. The Block Level is similar to the Record Level. However, at the Block Level, each request always reads or writes exactly one physical block of data instead of a user-defined quantity of data, as is true at the Record Level. In addition, data transfer is to and from a buffer provided by the Monitor, rather than a buffer provided by the user. The user may do his processing in the Monitor buffer or he may transfer data to his own area. Block Level requests may be used only with file-structured devices (i.e., disk and DECtape, but not magtape or cassette) and only with contiguous files.

To implement a BLOCK transfer, the programmer follows the sequence of requests shown in Figure 3-4b. Notice that the transfer must use .INIT, .OPEN, .WAIT, .CLOSE and .RLSE following the same rules as the RECORD level. The .BLOCK request has the address of the Link block and the Block block for its arguments.

The BLOCK block specifies the function (INPUT, GET, or OUTPUT), the relative number of the block being transferred to or from, the Monitor buffer address (supplied by the Monitor), and the length of the Monitor buffer (supplied by the Monitor). See Section 3-3.6.5.



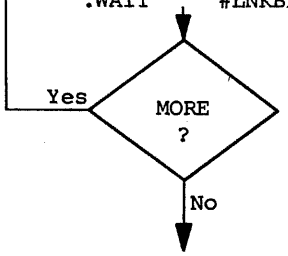
*Transfers between the Monitor's buffer and the user's buffer are optional and must be done by the user.

Figure 3-4a
The Transfer Path

```
        .INIT     #LNKBLK
        MOV       #FILBLK,R1
        .OPENU    #LNKBLK,R1
        .BLOCK    #LNKBLK,#BLKBLK      ;INPUT DESIRED BLOCK
        .WAIT     #LNKBLK              ;COULD BE REPLACED BY .WAITR

        (Process Data)                 ;UPDATE DATA

        .BLOCK    #LNKBLK,#BLKBLK      ;WRITE UPDATED BLOCK
        .WAIT     #LNKBLK
```




```
        .CLOSE    #LNKBLK
        .RLSE     #LNKBLK
        LNKBLK:             (entries)
        FILBLK:             (entries)
        BLKBLK:             (entries)
```

Figure 3-4b
The Sequence of Requests for .BLOCK

Figure 3-4
.BLOCK Input/Output Transfers

3.2.1.4 TRAN Level Requests

A TRAN level request is a basic input/output operation. No services are provided for the user other than to pass his request to the appropriate driver. The Monitor request .TRAN does not operate within a particular file structure as do .READ, .WRITE, .RECRD, and .BLOCK; hence no .OPEN or .CLOSE is used. Because .TRAN does not respect file structures, the user is strongly cautioned against using it for writing on file-structured devices, since he can easily do irreparable damage to information on such a device. Omitting the dataset logical name from the Link Block prevents another physical device from being assigned.

Data is transferred directly between the device and a buffer provided by the user (Figure 3-5a), with no formatting performed.

.TRAN is generally used in 2 situations:

1. When the file structure does not allow the desired operation (e.g., PIP uses .TRAN to read a directory block for the directory listing operation).

2. When one does not need or cannot afford the overhead of doing READ/WRITE processing on a non-file-structured device (e.g., a program to read data arriving at random intervals from an A/D converter might use .TRAN to read the data and .BLOCK to buffer the data on a disk for processing as time permits.

To implement a TRAN level I/O request, the programmer follows the sequence of macros shown in Figure 3-5b. Notice that the programmer must use .INIT and .RLSE, but must not use .OPEN or .CLOSE. The .TRAN request has the address of the TRAN Control Block (TRNBLK) as its argument. This block contains entries which specify the core starting address of the user's buffer, the device block address, the number of words to be transferred, and the function to be performed. .TRAN is therefore a device dependent request.

Table 3-5
Transfer Levels for Types of Datasets

Type of Transfer	Type of Dataset: Linked File	Contiguous File	Non-file-Structured Device
READ/WRITE	Yes	Yes	Yes
RECORD	No	Yes	No
BLOCK	No	Yes	No
TRAN	*	*	Yes

* indicates that TRAN may be used on a file-structured device if the warnings mentioned are observed. Usage in these cases is not advised.

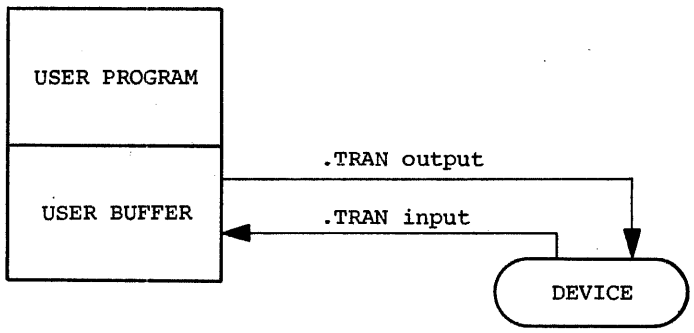


Figure 3-5a
The Transfer Path

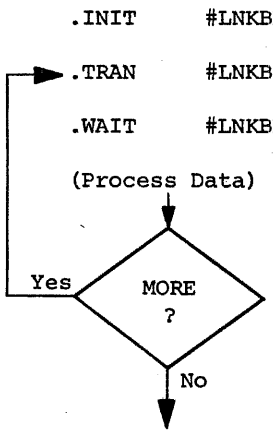


Figure 3-5b
Sample Sequence of Requests for .TRAN

Figure 3-5
.TRAN Input/Output Transfers

3.2.2 Requests for Directory Management Services

Directory management requests are used to enter filenames into directories, search for files, update filenames, and protect files against deletion. See Table 3-4 for the specific requests.

3.2.3 Requests for Miscellaneous Services

Requests for miscellaneous services include:

1. Requests to Load programs and overlays.
2. Requests to return control from a running program to the Monitor.
3. Requests to set Monitor parameters such as the TRAP vector or a program's restart address.
4. Requests to obtain Monitor parameters such as the size of the Monitor, the date, the time, and the current user's UIC.
5. Requests to perform conversions between ASCII and Radix-5Ø packed ASCII, binary and ASCII decimal, and binary and ASCII octal.
6. Requests to access the Command String Interpreter.
7. Requests to dump core.

See Table 3-4 for the list of miscellaneous service requests and their purposes.

3.3 DEVICE INDEPENDENCE

It is generally preferable to write programs so that each dataset may be associated with the widest possible variety of devices. This makes it easier to move a program from one configuration to another. It also makes it possible to use the program with a variety of different media. For example, the Assembler accepts input from disk, paper tape, DECtape, and other devices.

The Monitor makes it relatively easy to achieve this objective. Most I/O operations are completely device independent. No special actions by the user are required to accommodate the operation to the device. This holds true specifically for .READ, .WRITE, .OPEN, .CLOSE, .WAIT, .WAITR, .INIT, and .RLSE. In addition, .RECRD and .BLOCK require only that the device be file structured. Only .TRAN and .SPEC are typically device dependent.

In no case is a device associated with a dataset until an .INIT request is made. The device name may be specified in any of the following ways:

1. The programmer may specify the name in his Link Block.

2. The program can obtain a device name by requesting the user to enter a command string (Section 3-3.6.8); this will override any device specified in the Link Block.

3. The user can use the ASSIGN command (see Chapter 3-2) to associate a device (and file name) with the dataset; this option overrides both preceding options.

When a command string is requested by the program, it always overrides the link block specification. However, when ASSIGN is entered at the operator's discretion, it overrides the Link Block only when its logical dataset name is specified. In the latter case, it is best to supply a default physical device name in the Link Block.

Note that the substituted devices must be compatible. For example, the user may initially specify a BLOCK transfer from disk and later change the assignment to input from DECtape instead. But, he cannot later specify a paper tape reader as the input device, since BLOCK level requests are not usable on non-file-structured devices.

It is important to note that a device is assigned in a program to a dataset logical name and that reassigning a device at run time for one dataset logical name does not reassign that device for all dataset logical names to which it was originally assigned.

The only transfer requests which are not device independent are .TRAN and .SPEC.

3.4 OVERLAYING ROUTINES INTO CORE

Except for a small, permanently-resident portion, the Monitor routines which process most programmed requests are potentially swappable. They are normally disk resident and are swapped into core by the Monitor only when needed. The user may, however, specify that one or more of these potentially swappable routines be made permanently core resident or core resident only for the duration of his program's run.

Making a potentially swappable routine core resident ties up core space, but speeds up operation on the associated request. The user may, for example, be collecting data via a .TRAN request in a real-time environment. In such a case, even the short time needed to swap in the .TRAN request processor could cause him to lose data.

Any routine which services a programmed request other than .READ or .WRITE may be made core resident by one of the following methods:

1. Routines may be made permanently core resident at Monitor generation time (see the DOS/BATCH System Manager's Guide).

2. Routines may be made core resident for the duration of a program's run by declaring the appropriate global name (as specified in the definition of each request in Section 3-3.6) in a .GLOBL assembler directive in the user program. For example, to make the .TRAN processor resident while program FROP is being run, the following directive would be included in program FROP:

```
.GLOBL TRA.
```

Device drivers are loaded into the Monitor's free core area on an .INIT call and are freed from core on the occurrence of a .RLSE, provided no other dataset is INITed to that device.

3.5 MONITOR RESTRICTIONS ON THE USER

In return for the services provided by the Monitor, the programmer must honor certain restrictions:

1. The user should not use either the EMT or the IOT instructions for communication within his program.

2. It is recommended that the user not raise his interrupt priority level above 3, since it might lock out a device that is currently trying to do input/output, i.e., TTY interrupts at BR4.

3. HALT instructions are not recommended. If a HALT is executed during an I/O operation, most devices will stop, and only recovery from the console (pressing the CONTinue switch on the console) will be effective (recovery from the keyboard will not be immediately possible, since a HALT inhibits the keyboard interrupt). Some devices, such as DECtape, do not see the HALT and continue moving, thereby losing their positions over the block under transfer, and consequently can run the tape off the reel.

4. The RESET instruction should not be used because it forces a hardware reset: clearing all buffer registers and status flags, and disabling all interrupts, including keyboard interrupts. Since all I/O is interrupt driven, RESET will disable the system.

5. The user must not penetrate the Monitor when he is using the stack. The stack is set by the RUN time loader just below the lowest address of the program loaded. The Monitor checks to see that the stack is not overflowing each time it honors a request.

6. The user may allocate temporary storage areas on the stack by simply subtracting the size of the area needed from the current stack pointer value. When doing so, he should use a .MONF (Section 3-3.6.23) to determine the highest address being used by the Monitor. It is generally wise to leave some space for future Monitor expansion (as a consequence of programmed requests) and for stack extension (as a consequence of subroutine calls, Monitor requests, device interrupts, etc.). Consult Figure 3-6 for more information about Monitor core usage.

7. The user should be aware that certain requests, such as .INIT, may change the amount of available free core, since the request may call in drivers and establish data blocks. Such requests affect the result of .MONF requests.

8. Certain requests return data to the user on the stack. The user must clear this data from the stack before the stack is used again. The Monitor clears the stack after it honors requests that do not return data to the user on the stack.

9. The user should not use global names that are listed in Chapter 3-9.

10. The Link pointer in the Link Block is set by the Monitor and must not be altered by the user.

11. The stack should be left at the bottom of the user's core area to ensure protection against the Monitor buffer area's overlaying the user's area.

12. The Monitor uses the stack (via R6) for communication to the user and for its own functions; the user should therefore not use R6 for arithmetic operations as the Monitor might need the stack while it was corrupted.

3.6 REQUEST FOR MONITOR SERVICES

.ALLOC

3.6.1 .ALLOC - Allocate (create a contiguous file).

Macro Call: .ALLOC #LNKBLK,#FILBLK,#N

where LNKBLK is the address of the Link Block, FILBLK is the address of the Filename Block, and N is the number of 64-word segments requested.

Assembly Language Expansion:

```
MOV #N,-(SP) or MOV #N+1ØØØØØ,-(SP)
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 15
```

Global Name: ALO. (See Appendix C for subsidiary routines.)

Description: Searches the device for a free area equal to N 64-word segments, and creates a contiguous file in the area if it is found, by making an appropriate entry in the User File Directory (UFD). If the sign bit (bit 15) of N is set, the UFD pointer will point to the beginning of the allocated area thereby indicating that the file is empty. This enables partial filling of the file space and later extension of the file. If the sign bit of N is not set, the UFD pointer will point to the end of the allocated area and thereby indicate that the file area is full and may not later be extended. (Linked files are created by an .OPENO request.) The search begins at the high end of the device. The number of blocks allocated will be the minimum number required to contain N segments, i.e.,

$$\left\lceil \frac{N}{B} \right\rceil$$

where B is the number of 64-word segments per block. For example, if N=9 and the device specified is DECtape, then $B=\frac{256}{64} = 4$. Therefore, $\frac{N}{B} = \frac{9}{4} = 3$, and 3 blocks will be allocated.

After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack, and the top word of the stack will be set to -1 to indicate the successful completion of the request, or to the largest number of segments currently available if this is less than the called request. The value will be meaningless if the call cannot be met because of any other error.

If a 64-word segment count of zero is detected, the next word on the stack is assumed to be the number of device standard size blocks to be allocated. The sign bit on the zero count retains the same meaning as the standard request. The device standard size count remains on the stack after the call.

This type of call allows allocation of large RPØ3 files that cannot be allocated in the normal manner. This version is not recommended for any other use.

Rules: Must be preceded by an .INIT request on the dataset. A Filename Block must be set up by the user in his program.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or to the terminal for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	AØØ2
Dataset Not INITed	--	FØØØ
File Exists	2	FØ24
Directory Full	12	FØ24
UIC Not In Directory	13	FØ24
Illegal Filename	15	FØ24

If the error address in the Filename Block is taken, the top word of the stack is meaningless.

Example: Create a contiguous file of four 256_{10} word blocks on DECtape unit 4. Name the file FREQ.DAT.

```
            .
            .
            .
            .ALLOC   #FRQ,#FREQIN,#2Ø     ;ALLOCATE A DATASET
            INC      @SP                  ;CALL COMPLETE?
            BNE      NOROOM               ;  NO
            .
            .
            .
            .WORD    ERR1                 ;LINK BLOCK
FRQ:        .WORD    Ø
```

```
            .RAD5Ø   /DTA/
            .BYTE    1,4
            .RAD5Ø   /DT/
               :
               :
            .WORD    ERR2
            .WORD    Ø
FREQIN:     .RAD5Ø   /FRE/
            .RAD5Ø   /Q/
            .RAD5Ø   /DAT/
            .WORD    UIC,PROT1
               :
               :
ERR1:                                ;TO HERE IF NO BUFFER AVAILABLE
               :                     ;FOR DRIVER
               :
ERR2:                                ;TO HERE IF FILE STRUCTURED ERROR
               :
               :
NOROOM:                              ;TO HERE IF NOT ENOUGH CONTIGUOUS
               :                     ;BLOCKS ON DEVICE
               :
```

.APPND 3.6.2 .APPND - Append one linked file to another.

Macro Call: .APPND #LNKBLK,#FIRST,#SECOND
where LNKBLK is the address of the Link Block, FIRST is the address of the Filename Block for the first file (file to be appended to), and SECOND is the address of the Filename Block for the second file (file to be appended).

Assembly Language Expansion:

```
MOV #SECOND,-(SP)
MOV #FIRST,-(SP)
MOV #LNKBLK,-(SP)
EMT 22
```

Global Name: APP. (See Chapter 3-5 for subsidiary routines.)

Description: Makes one linked file out of two by appending the SECOND to the FIRST. The directory entry of the SECOND file is deleted. When the request is completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack. No attempt is made to pack the two files together, the physical blocks are merely linked together.

Since the last block of a file is typically not full, there will be a gap (null characters) in the new file at the junction point. This causes no problem in ASCII files but might cause confusion in binary files.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the offending Filename Block if it is specified, or to the terminal for an error message if it is not. Possible errors are illustrated on the following page.

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	AØØ2
Dataset Not INITed	--	FØØØ
First File Nonexistent	2	FØ24
Contiguous File	5	FØ24
Protect Code Violated	6	FØ24
File Opened	14	FØ24

3.6.3 .BIN2D - Convert one binary word into five decimal ASCII characters. **.BIN2D**

Macro Call: .BIN2D #ADDR,#WORD
where ADDR is the address of the first byte of the buffer where the characters are to be placed, and WORD is the number to be converted.

Assembly Language Expansion:

```
MOV #WORD,-(SP)
MOV #ADDR,-(SP)
MOV #3,-(SP)                ;MOVE CALL CODE ONTO STACK
EMT 42
```

Global Name: CVT.

Description: WORD is converted into a string of five decimal 7-bit ASCII characters which are placed into consecutive bytes starting at location ADDR. They are right-justified with leading zeros. The stack is cleared.

3.6.4 .BIN2O - Convert one binary word into six octal ASCII characters. **.BIN2O**

Macro Call: .BIN2O #ADDR,#WORD
where ADDR is the address of the first byte of the buffer into which the six octal ASCII characters are to be placed, and WORD is the binary number to be converted.

Assembly Language Expansion:

```
MOV #WORD,-(SP)
MOV #ADDR,-(SP)
MOV #5,-(SP)
EMT 42
```

Global Name: CVT.

Description: The WORD is converted into a 6-byte string of 7-bit octal ASCII characters, right-justified with leading zeros, which is placed into the buffer addressed by ADDR. The stack is cleared.

.BLOCK 3.6.5 .BLOCK - Read or write a specific block in a file.

Macro Call: .BLOCK #LNKBLK,#BLKBLK

where LNKBLK is the address of the Link Block, and BLKBLK is the address of the BLOCK block (see Figure 3-13).

Assembly Language Expansion:

```
MOV #BLKBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 11
```

Global Name: BLO.

Description: BLOCK requests provide for random access to the blocks of files stored on disk or DECtape. This function is not supported for magtape or cassette.

In this mode, data is transmitted to or from a specified block in a file with no formatting performed. Transfers take place between the device block and a Monitor buffer. The user may process the data in the Monitor buffer or he may transfer the block to and from his own area. BLOCK requests require the use of the .INIT, .OPEN, .CLOSE and .WAIT (or .WAITR) requests.

The user must specify one of three functions in the BLOCK block: INPUT, GET, or OUTPUT (see Figure 3-13). After the transfer has started, control is returned to the user at the instruction following the assembly language expansion with arguments removed from the stack.

INPUT: During an INPUT request, the requested block of the requested file is read into a Monitor buffer, and the user is given in the BLOCK block (see Figure 3-13) the address of the buffer and the physical length of the block transferred.

GET: During a GET request, the Monitor returns in the BLOCK Block the address and length of a buffer within the Monitor that he can fill for subsequent output. Only one GET is required for each time the file is OPENed and CLOSEd (i.e., once a buffer has been located, it may be used repeatedly). The user must assure that he does not over-run the buffer. This request is unnecessary if an INPUT request has occurred.

OUTPUT: During an OUTPUT request, the contents of the buffer assigned is written on the device in the requested relative position in the requested file.

Rules: The associated file must be opened by .OPENI for input or .OPENU for input or output.

Access to linked files or nondirectory devices is illegal.

The user must set up the BLOCK block in his program according to the format of Figure 3-13.

Errors: Error processing causes a normal return to the user, with the type of error indicated in the FUNCTION/STATUS word of the BLOCK block. The user should perform

```
TSTB BLKBLK+1
BNE ERROR
```

after a .WAIT to assure that his request was error free.

3.6.6 .CLOSE - Close a dataset. .CLOSE

Macro Call: .CLOSE #LNKBLK

where LNKBLK is the address of the Link Block (see Figure 3-7).

Assembly Language Expansion:

```
MOV #LNKBLK,-(SP)
EMT 17
```

Global Name: CLS. (See Chapter 3-5 for subsidiary routines.)

Description: The .CLOSE request indicates to the Monitor that no more I/O requests will be made on the dataset. .CLOSE completes any outstanding processing on the dataset (e.g., on output, it writes the last buffer; on extension, it links the extension to the old file; etc.), updates any directories affected by the processing, and releases to free core any buffer space established for the processing. When a file which has been opened for output is closed, the last block written and the last byte written are recorded in the directory to indicate end-of-data. This eliminates the need to pad out blocks with nulls and allows the written data within a contiguous file to be extended at a later time.

After the .CLOSE request has been completed, control is returned to the user at the instruction following the assembly language expansion; the argument is removed from the stack. As with .OPEN, some appropriate device action may still be in progress at this point.

Rules: The dataset to be closed must have previously been opened if it was a file on a file-structured device.

As with .OPENx, a .CLOSE is not required if the dataset is not a file, but it is strongly recommended in order to maintain device independence.

Errors: Dataset Not INITed - Fatal Error FØØØ;
Device Parity Error - Fatal Error FØ17

All error messages are explained in Appendix K.

Example: Open for input a dataset named IMP, which is file PROG1.BIN on DECtape unit 3. After the data transfer is complete, close the file.

```
                .INIT #SET1
                   :
                .OPEN #SET1,#FILE1        ;OPEN SET1 FOR INPUT (OPEN CODE
                   :                      ;IS IN FILE BLOCK)
                (Input is
                 Performed
                 Here)
                   :
                .CLOSE #SET1              ;CLOSE SET1
                   :
                .RLSE  #SET1
                   :
                .WORD  ERR1
SET1:           .WORD  Ø
                .RAD5Ø /IMP/              ;DATASET NAME
                .BYTE  1,3
                .RAD5Ø /DT/               ;PHYSICAL DEVICE NAME
                   :
                .WORD  ERF1               ;ADDR OF ERROR RTN
                .WORD  4                  ;OPEN FOR INPUT
FILE1:          .RAD5Ø /PRO/              ;FILENAME
                .RAD5Ø /G1/
                .RAD5Ø /BIN/              ;EXTENSION
                .BYTE  PROG,PROJ
                .BYTE  177
                .EVEN
                   :
ERR1:                                     ;HERE FOR .INIT, .OPENI, .CLOSE,
                   :                      ;OR .RLSE ERRORS (DEVICE)
ERF1:                                     ;HERE FOR .OPENI ERRORS
                   :                      ;(DATA FILE)
```

.CORE 3.6.7 .CORE - Obtain address of the highest word in core memory.

Macro Call: .CORE

Assembly Language Expansion:

```
MOV #1ØØ,-(SP)      ;CODE
EMT 41
```

Global Name: GUT.

Description: Determines the address of the highest word in core memory (core size minus 2) and returns it on the top of the stack. For an 8K machine, it would return 37776. The user must clear the stack.

3.6.8 Requests for Interfacing with the Command String Interpreter

A user program may obtain dataset specifications via keyboard input at run time by calling the Command String Interpreter (CSI) routine. This routine is used by many system programs; it accepts keyboard input at program run time in the format presented in Appendix H.

The CSI is called in two parts, by two different requests:

.CSI1	condenses the command string and checks for syntactical errors.
.CSI2	sets the appropriate Link Block and Filename Block parameters for each dataset specification in the command string.

Each command string requires one .CSI1 request for the entire command string, and one .CSI2 request for each dataset specifier in the command string.

The user must first set up a line buffer in his program and read in the command string (see Section 3-4.3). Then he does a .CSI1, which condenses the string by eliminating spaces, horizontal TABs, nulls, and RUBOUTs, sets pointers in a table to be referenced by .CSI2, and checks the command string for syntactical errors. If there are no errors, the .CSI2 request may be given once for each dataset specification that the user expects to find in the command string. .CSI2 fills in the appropriate Link Block and Filename Block parameters according to the device name, filename, extension, UIC, and switch entries in the command string.

3.6.8.1 .CSI1 - Condense command string and check syntax. **.CSI1**

Macro Call: .CSI1 #CMDBUF
where CMDBUF is the address of the command buffer header described under "Rules" below.

Assembly Language Expansion:

```
MOV #CMDBUF,-(SP)
EMT 56
```

Global Name: CSX.

Description: Condenses the command string by removing spaces, horizontal TABs, nulls, and RUBOUTs, and checks the entire command string for syntactical errors. Control is returned to the user with a Ø at the top of the stack if the syntax is acceptable, or with the address (in the command string line buffer) of the data byte at which the scan terminated because the first error was encountered.

If .CSI1 encounters a CTRL/C in the command string, it executes an immediate .EXIT EMT thus terminating program execution. CTRL/C can be entered in a command string by immediately preceding it with an ALTmode character.

Rules: The .CSI2 request must be preceded by a .CSI1 request, because .CSI2 assumes it will get a syntactically correct command; more than one .CSI2 request can follow a single .CSI1 request.

The user must set up a line buffer and read in the command string before doing .CSI1. Command Strings must not be read in dump mode.

It is the user's responsibility to print a # on the terminal to inform the operator that a CSI format is expected (Section 3-2.1). If VERTICAL TAB is used as the terminator, the # will be typed immediately without a carriage return or line feed.

The user must set up a seven-word work area (CMDBUF) in his program immediately preceding the header of the line buffer into which the command is to be read. The user is not required at this time to set up anything in the work area (CMDBUF) prior to calling .CSI1; it will be used as a work-and-communication area by the Monitor routines which process the .CSI1 and .CSI2 requests.

The user must clear the stack upon return from the Monitor. If the top of the stack ≠ Ø (i.e., if there was a syntax error), .CSI2 must not be called.

Example: See .CSI2, Section 3-3.6.8.2.

.CSI2

3.6.8.2 .CSI2 - Interpret one dataset specification of a command string.

Macro Call: .CSI2 #CSIBLK
where CSIBLK is the CSI control block, described under "Rules" below.

Assembly Language Expansion:

```
MOV #CSIBLK,-(SP)
EMT 57
```

Global Name: CSM.

Description: Gets the next input or output dataset specification from the command string, and sets the PHYSICAL DEVICE NAME entry in the Link Block, the FILENAME, EXTENSION, and UIC entries in the Filename Block, and any switch entries in an extension of the Link Block.

Rules: Before calling .CSI2, the user must:

- Call .CSI1 to condense the command string and check it for syntax errors. There must have been no syntax errors.
- Set up a CSI control block as follows:

CSIBLK:

POINTER TO CMDBUF
POINTER TO LNKBLK
POINTER TO FILBLK

where POINTER TO CMDBUF is the address of the 7-word work area preceding the command string line buffer header;

POINTER TO LNKBLK is the address of the Link Block of the dataset whose specification is being requested; and

POINTER TO FILBLK is the address of the Filename Block of the dataset whose specification is being requested (currently, CSI allows only one file per dataset specification).

- Set the first word (Code Word) of CMDBUF to either Ø or 2. Ø means "get next input dataset specification", and 2 means "get the next output dataset specification". .CSI2 does not check the validity of the code word.
- Initialize the NUMBER OF WORDS TO FOLLOW entry in the Link Block to contain the number of words to follow. This must be at least one, because .CSI2 will alter the following word, i.e., the PHYSICAL DEVICE NAME word. .CSI2 does not check the validity of this byte.

  The user may specify any number from 1 to 255_{10} in this location. All words in excess of 1 are used for switch space.

Upon return from the .CSI2 request, the Monitor will have provided the following information:

1. The top of the stack contains two items of information. Bits 1-Ø have the following meaning:
   a. Ø, which means the dataset specification requested has been obtained, and there are still more dataset specifications of the type requested (i.e., input or output); or
   b. 1, which means the dataset specification requested has been obtained, and there are no further dataset specifications of the type requested; or
   c. 2, which means (a), but this particular dataset specification included more switches than would fit in the space provided; or

d. 3, which means (b), but this particular dataset specification included more switches than would fit in the space provided.

If there are no more dataset specifications and the user requests one anyway, a null specification will be returned.

Bit 2, when set to one, indicates that the device name in the Link Block is a default supplied by the system (see Section 3-6.2).

2. With respect to values returned in the Link Block (Figure 3-7):

If the PHYSICAL DEVICE NAME word is zero, the user does not wish this particular output (input) dataset to be generated (read); i.e., this entry was omitted when the command string was typed. If not zero, the PHYSICAL DEVICE NAME and UNIT NUMBER are appropriately set to the device and unit specified in the command string.

3. Immediately following the PHYSICAL DEVICE NAME word in the Link Block are the switches specified in the command string. The interface for each switch is shown in the switch block below. These switch blocks are written in the area provided by the programmer in the Link Block. Note that the number of words to follow in the switch block is not the same quantity as is specified in the LINK Block

LNKBLK+1Ø:

NUMBER OF WORDS TO FOLLOW	
POINTER TO FIRST CHARACTER OF Vn	
POINTER TO FIRST CHARACTER OF Vn-1	
⋮	
POINTER TO FIRST CHARACTER OF V1	
W(ASCII)	S(ASCII)

;for /SW

If NUMBER OF WORDS TO FOLLOW is zero, there are no more switches. Note that the pointers are in reverse order. After the value pointers are the ASCII bytes which contain the first two characters of the switch. The first character is in the low byte, and the second is in the high byte. If the name of the switch contains only one character, the ASCII representation of that character will be in the low byte, and the high byte will contain a zero. Note that if the NUMBER OF WORDS TO FOLLOW is not zero, it is the number of values +1. For example, if the switch /SWITCH:$12:AB is stored in memory beginning at location 1ØØØ as:

1ØØØ	1ØØ1	1ØØ2	1ØØ3	1ØØ4	1ØØ5	1ØØ6
/	S	W	I	T	C	H
1ØØ7	1Ø1Ø	1Ø11	1Ø12	1Ø13	1Ø14	1Ø15
:	$	1	2	:	A	B

then the completed interface appears as:

LNKBLK+1Ø:

3	
1Ø14	
1Ø1Ø	
127=W	123=S

With respect to the values returned in the Filename Block (Figure 3-8):

a. The FILENAME occupies the two words at FILBLK and FILBLK+2. If the Monitor returns zero at FILBLK, no filename was specified in the dataset specification; if it returns 52_8 at FILBLK, * was specified as the filename. Otherwise, the Monitor returns at FILBLK and FILBLK+2 the first six characters of the filename specified, in Radix-5Ø packed ASCII.

b. The EXTENSION occupies the word at FILBLK+4. If the Monitor returns zero at FILBLK+4, no extension was specified; if it returns 52_8, * was specified. Otherwise, the Monitor returns the first three characters of the extension specified, in Radix-5Ø packed ASCII.

   The system does not distinguish between a name with no extension and a name with an extension of all blanks. E.g., NAME and NAME.ΔΔΔ are equivalent.

c. The USER IDENTIFICATION CODE occupies the word at FILBLK+6. If the Monitor returns zero at FILBLK+6, no UIC was specified in the dataset specification (the I/O processors will assume the UIC of this user). If a UIC was typed in, the Monitor will set this word appropriately. The Monitor returns 377_8 in the high- or low-order byte of this word if * was specified in either of those positions.

   The user may restart at the beginning of the input dataset or output dataset side of the command string simply by recalling .CSI1 and issuing a Ø or 2 code, respectively. Note that he may not restart one without restarting the other.

Remark: There is no error checking with respect to magnitude when the UNIT or UIC values are converted from octal ASCII to binary.

3.6.9 .CVTDT - Convert binary representation of date or time to ASCII character string. .CVTDT

Macro Call: .CVTDT #CODE, #ADDR[,VALUE]

where CODE identifies the conversion to be done;

```
CODE = Ø    Current date as stored by monitor,
CODE = 1    Current time as stored by monitor,
CODE = 2    Date supplied as VALUE,
CODE = 3    Time supplied as VALUE (and VALUE+2)
```

ADDR is the address of the first byte of the user buffer into which the ASCII string is to be stored, and VALUE is the address of user supplied Date or Time (used with CODEs 2 and 3 only).

Assembly Language Expansion:

```
MOV VALUE+2,-(SP)          ;Code 3 only
MOV VALUE,-(SP)            ;Codes 2 and 3 only
MOV #ADDR,-(SP)
MOV #CODE,-(SP)
EMT 66
```

Global Name: CDT.

Description: This request converts either a date or a time from internal (binary) representation into an ASCII string suitable for display. The user may specify that the current system value (of date or time) is to be used for conversion or he may supply his own value. The string returned has the format of the Date and Time returned by the Keyboard DATE and TIME commands (see Chapter 3-2). Upon return, the call arguments have been removed from the stack and condition codes N, Z and V are cleared to Ø. If .CVTDT is called with a code of 1 and the time is past midnight (i.e., 24:ØØ), a .CVTDT will automatically update the time and date as stored in the Monitor, before supplying the converted value to the calling program.

Rules:

1. The buffer area supplied by the user program (starting at ADDR) must provide sufficient room for the text returned as no check can be made. Nine bytes are required for Date, eight bytes are required for Time.

2. User-supplied VALUEs for Date or Time must comply with the internal storage format of those values, that is:

   a. Date; 1 word containing (year-197Ø)*1ØØØ + day of the year (Julian).

   b. Time; 2 unsigned integer words for high-order and low-order time in clock ticks.

Errors:

1. Specification of an illegal CODE (i.e., > 3) causes fatal error message:

   FØ34 Call address

2. If the currently stored date or time is out of range (i.e., date > 366 (Modulo 1ØØØ) or time > 47:59:59), an operator action message

   AØ11 CODE(Ø = Date, 1 = Time)

   is printed. The operator should enter the desired value via the appropriate DAte or TIme keyboard command and type COntinue to proceed. If 23:59:59 < Time < 48:ØØ:ØØ, the date is incremented and the time is reduced by 24:ØØ:ØØ.

3. If a user supplied date or time is out of range as above, the conversion routine will return without attempting conversion and the condition code V will be set to 1. Thus the program should follow the .CVTDT request with the check:

   BVS (error routine).

.DATE 3.6.10 .DATE - Obtain current date.

Macro Call: .DATE

Assembly Language Expansion:

```
MOV #1Ø3,-(SP)
EMT 41
```

Global Name: GUT.

Description: The current date word is returned to the user at the top of the stack. The user must clear the stack. The date format is a binary number equal to (year-197Ø)*1ØØØ$_{1Ø}$+day (Julian). If the user requires the ASCII representation of the date, he should use the .CVTDT request.

3.6.11 .DELET - Delete a file. .DELET

Macro Call: .DELET #LNKBLK,#FILBLK
where LNKBLK is the address of the Link Block, and FILBLK is the address of the Filename Block.

Assembly Language Expansion:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 21
```

Global Name: DEL. (See Chapter 3-5 for subsidiary routines.)

Description: Deletes from directory-oriented device the file named in the Filename Block. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack.

Rules: .DELET operates on both contiguous and linked files. It must be preceded by an .INIT on the dataset. If the file has been OPENed, it must be CLOSEd before it is deleted.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or to the terminal for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	AØØ2
Dataset Not INITed	--	FØØØ
Nonexistent File	2	FØ24
Protect Code Violation	6	FØ24
File Is Open	14	FØ24

.D2BIN 3.6.12 .D2BIN - Convert five decimal ASCII characters into one binary word.

Macro Call: .D2BIN #ADDR

where ADDR is the address of the first byte in the 5-byte string of decimal characters to be converted.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #2,-(SP)              ;MOVE CALL CODE ONTO STACK
EMT 42
```

Global Name: CVT.

Description: The 5-byte string of 7- or 8-bit decimal ASCII characters which start at ADDR are converted into their binary equivalent. The converted value is returned to the top of the stack, right-justified, followed by the address of the byte which follows the last character converted. The largest decimal number that can be converted is 65,535 (2^{16}-1). The user must clear the stack.

Errors: The conversion will be stopped if an error condition is encountered. The user will be informed of the type of error via the condition codes in the Processor Status register.

C-bit set means that a byte was not a decimal digit.
V-bit set means that the decimal number was too large, i.e., greater than 65535.

The value returned will be correct up to the last valid byte. The address returned will be that of the invalid byte. If the conversion is satisfactory, the condition codes will be cleared.

.EXIT 3.6.13 .EXIT - Exit from a user program to Monitor.

Macro Call: .EXIT

Assembly Language Expansion:

```
EMT 6Ø
```

Global Name: XIT.

Description: This is the last statement executed in a user's program. It returns control to the Monitor, assures that all of the program's data files have been closed and, in general, prepares for the next keyboard request. After the exit, all Monitor buffer space reserved for the program, such as Device Assignment Tables (DAT) established during program execution, are returned to free core.

3.6.14 .GTCIL - Return the address of the first block of the Monitor core image library (CIL). **.GTCIL**

Macro Call: .GTCIL

Assembly Language Expansion:

```
MOV #111,-(SP)
EMT 41
```

Global Name: GUT.

Description: This request returns the address of the first block of the Monitor core image library to the top of the stack.

Rules: The user is required to clear the disk address returned on the stack.

3.6.15 .GTCLK - Obtains system clock information. **.GTCLK**

Macro Call: .GTCLK

Assembly Language Expansion:

```
MOV #113,-(SP)
EMT 41
```

Global Name: GUT.

Returns: Bits Ø, 6, and 7 or (SP) are used to indicate the following:

Bit Ø = Ø to indicate 60 hz.
1 to indicate 50 hz.

Bit 6 = Ø to indicate KW11L.
1 to indicate KW11P.

Bit 7 = Ø to indicate no clock.
1 to indicate that there is a clock.

NOTE

If both KW11L and KW11P are present, the KW11L is used as the system clock.

Description: .GTCLK obtains information about the system clock and returns it on the stack.

Rules: The user must clear the stack.

.GTOVF 3.6.16 .GTOVF - Obtains and sets the overlay flag.

Macro Call: .GTOVF

Assembly Language Expansion:

```
MOV #144,-(SP)
EMT 41
```

Returns: (SP) = address of the variable OVLFLG in the SVT.

Description: .GTOVFL sets the value of the variable OVLFLG to one. It is reset to zero by either of the following conditions:

1. Execution of a KILL or BEGIN Monitor command,
2. The exit EMT.

.GTOVF is used by the overlay system. It should not be used in overlayed programs.

Rules: The user must clear the stack.

.GTPLA 3.6.17 .GTPLA - Return the current program low address.

Macro Call: .GTPLA

Assembly Language Expansion:

```
CLR -(SP)
MOV #5,-(SP)
EMT 41
```

Global Name: GUT.

Description: The program's low address is the address of the first (lowest) word of the current program. In the case of a program with overlays, the PLA is the address of the first word of the resident section. PLA is established when the keyboard RUN command is executed or when the .RUN request is used to load a new program (not an overlay, e.g., when MACRO calls CREF, which then replaces MACRO). Because the .RUN processor will not load an overlay which extends above this address, the PLA is also called the Protection Boundary.

.GTPLA allows the user to retrieve this value (see Figure 3-6), which is returned to the top of the stack. The .STPLA request allows the user to set it.

Rules: The user must clear the stack.

3.6.18 .GTRDV - Gets run device information. .GTRDV

Macro Call: .GTRDV

Assembly Language Expansion:

```
CLR   -(SP)
CLR   -(SP)
MOV   #112,-(SP)
EMT   41
```

Returns:
(SP) = Device name from last RUN EMT call (RAD5Ø).
(SP+2) = Unit number in bits 2 through Ø.
(SP+4) = Starting block number if the file was contiguous and on a file-structured device; otherwise it is Ø.

Description: .GTRDV obtains information about the last run device and places it in the indicated location.

Rules: The user must clear arguments from the stack.

3.6.19 .GTSTK - Return the current stack base entry. .GTSTK

Macro Call: .GTSTK

Assembly Language Expansion:

```
CLR -(SP)
MOV #4,-(SP)
EMT 41
```

Global Name: GUT.

Description: The stack base is the highest core address used for stack storage plus two. A RUN keyboard command clears the stack and sets the stack base address to the program low address. A user .RUN request does not clear the stack (to allow inter-program communication via the stack) but the stack may be relocated. This request may be used to determine the stack base. Following the request the current stack base entry is returned on top of the stack.

Rules: The user is required to clear the returned value from the stack.

.GTUIC 3.6.20 .GTUIC - Get the current user's UIC.

Macro Call: .GTUIC

Assembly Language Expansion:

```
MOV #1Ø5,-(SP)        ;CODE
EMT 41
```

Global Name: GUT.

Description: The current user's UIC is returned at the top of the stack in the form:

GROUP NUMBER	USER'S NUMBER
HIGH-ORDER BYTE	LOW-ORDER BYTE

Rules: The user must clear the stack.

.INIT 3.6.21 .INIT - Associate a dataset with a device driver and set up the initial linkage.

Macro Call: .INIT #LNKBLK
where LNKBLK is the address of the Link Block.

Assembly Language Expansion:

```
MOV #LNKBLK,-(SP)
EMT 6
```

Global Name: INR.

Description: Assigns a device to a dataset and assures that the appropriate driver exists and is in core. If the driver is not in core, it is loaded. The device assigned is that specified in the associated Link Block, unless assignment has been made to the logical name specified in the Link Block with the ASSIGN command or via the Command String Interpreter. After the .INIT has been completed, control is returned to the user at the instruction following the assembly language expansion. The argument is removed from the stack.

Rules: The user must set up within his program a Link Block of the format explained in Section 3-4.1 for each dataset to be .INITED. A dataset which has been .INITed should be .RLSEd prior to any further .INIT request for that Link Block.

Errors: A nonfatal error message, AØØ3, is printed on the terminal if no assignment has been made through the ASSIGN command, and the DEFAULT DEVICE is either not specified in the Link Block or has been specified illegally (i.e., no such device on the system). The user may type in an assignment (ASSIGN) and give the CONTINUE console command to resume operation.

The AØØ3 error message also is issued by .INIT if an attempt is made to assign a Link Block to a device that already has an outstanding Link Block and whose driver does not support more than one user.

Control is transferred to the address specified by the error return address in the Link Block if at any time during an operation there is not enough space in free core for the necessary drivers, buffers, or tables. If no address (i.e., a zero) is specified in the Link Block's ERROR RETURN ADDRESS, a fatal (FØØ7) error is printed and the program stops.

Example: See the .RLSE request.

3.6.22 .LOOK - Search the device directory for a specified filename. .LOOK

Macro Call: .LOOK #LNKBLK,#FILBLK[,1]
where LNKBLK is the address of the Link Block, and FILBLK is the address of the Filename Block.

Assembly Language Expansion:

a. If the optional argument is not specified:

```
MOV #FILBLK,-(SP)
MOV # LNKBLK,-(SP)
EMT 14
```

b. If the optional argument is specified:

```
MOV #FILBLK,-(SP)
CLR -(SP)
MOV #LNKBLK,-(SP)
EMT 14
```

Global Name: DIR. (See Chapter 3-5 for subsidiary routines.)

Description: The primary purpose of this routine is to search through a specified directory for a specified file and return with the current parameters of the file. However, this routine can also be used to indicate (bits Ø-3) the permissible functions for a nondirectory device (i.e., input, output, update, etc.). By specifying the optional argument, the user indicates whether he requires two or three parameters be returned.

The device to be searched is specified in the Link Block, and the file is specified in the Filename Block. The request returns to the user with the top elements of the stack as follows:

	2 Arg. Call	3 Arg. Call
START BLOCK		SP
# OF BLOCKS	SP	SP+2
INDICATOR WORD	SP+2	SP+4

where # OF BLOCKS is the number of blocks in the file, and the INDICATOR WORD is coded as follows:

Bit Ø=1	.OPENC allowed
Bit 1=1	.OPENI allowed
Bit 2=1	.OPENE allowed
Bit 3=1	.OPENU allowed
Bit 4=Ø	File is not in use
4=1	File is being used by another dataset
Bit 5=1	Dataset already has a file open (no search has been performed)
Bit 6=Ø	File is linked
6=1	File is contiguous
Bit 7=Ø	File nonexistent (.OPENO allowed)
7=1	File exists or .OPENO not allowed
Bits 8-15	Protection Code

After the request has been completed, control is returned to the user at the instruction following the assembly expansion. The stack must be cleared by the user. If a file is protected against READ access, it will be signaled as non-existent.

Rules: The dataset must be INITed.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or to the terminal for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message
Device Not Ready	--	AØØ2
A File Is Open On Requesting Dataset	14	FØ24
Illegal Filename	15	FØ24

Note that it is possible to .LOOK for a file and be told that it does not exist. A subsequent attempt to open the nonexistent file may lead to an OPEN error (code=2). Hence, it may be more efficient to simply attempt the .OPEN request and check for an error.

3.6.23 .MONF - Obtain the address of the first word above the Monitor's highest allocated free core buffer. **.MONF**

Macro Call: .MONF

Assembly Language Expansion:

```
MOV #1Ø2,-(SP)
EMT 41
```

Global Name: GUT.

Description: The address of the first word above total Monitor area (see Figure 3-6), including the buffer and transient areas current at the time of the request, is returned to the user at the top of the stack. After the request is completed, control is returned to the user at the instruction following the assembly language expansion.

Rules: The user must clear the stack. Since buffers are allocated by the Monitor in its processing of certain requests, .MONF should be requested in the program at the point where the information is actually required.

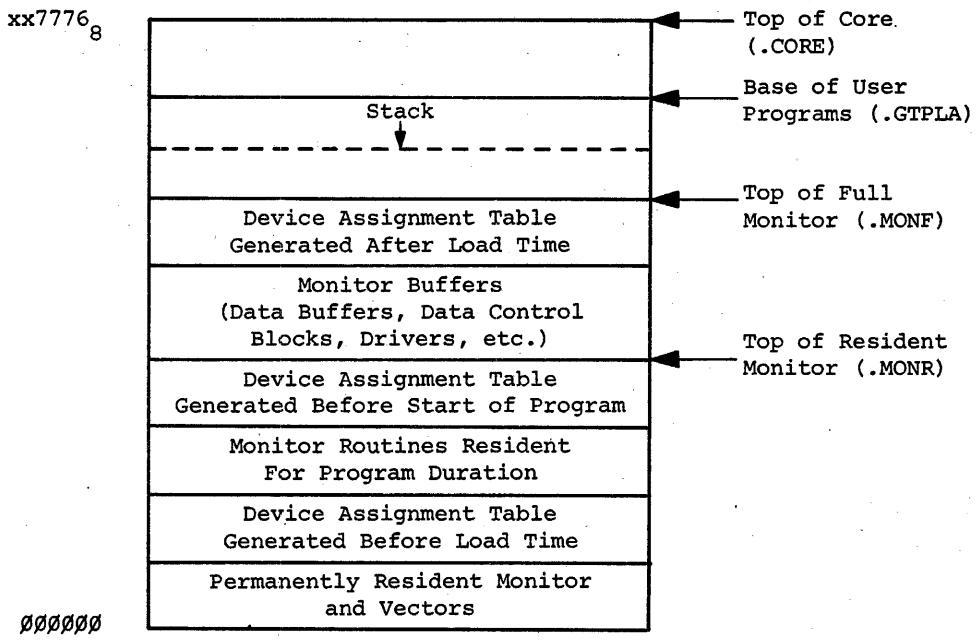


Figure 3-6
Core Map of Resident Monitor and Full Monitor

.MONR

3.6.24 .MONR - Obtain the address of the first word not within the resident Monitor.

Macro Call: .MONR

Assembly Language Expansion:

```
MOV #1Ø1,-(SP)
EMT 41
```

Global Name: GUT.

Description: Determines the first word above the top of the currently resident Monitor (see Figure 3-6) and returns it to the user at the top of the stack. This value does not reflect any area allocated by the Monitor for control blocks, device drivers, data buffers, etc. (see .MONF, Section 3-3.6.23). After the request is completed, control is returned to the user at the instruction following the assembly language expansion.

Rules: The user must clear the stack.

.OPEN

3.6.25 .OPEN - Prepare a device (which has been .INITed) for data transfer and associate the dataset with a file (if the device is file-structured).

Macro Call: .OPEN #LNKBLK,#FILBLK

This form assumes that the File Block contains a code indicating how the file is to be opened (see Description below).

Assembly Language Expansion:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 16
```

Alternate Form of Macro Call:

```
.OPENx #LNKBLK,Rn
```

where Rn is a register containing the address of the File Block and x indicates the type of .OPEN (see Description below).

Assembly Language Expansion:

```
MOVB #CODE,-2(Rn)      (see Description below)
MOV  Rn,-(SP)
MOV  #LNKBLK,-(SP)
EMT 16
```

Global Name: OPN. (See Chapter 3-5 for subsidiary routines.)

Description: When used, .OPEN follows .INIT or .CLOSE if more than one file is to be opened on the same LINK block. When the device being used is file-structured, .OPEN associates a specific file with the dataset. .OPEN also acquires a data buffer and prepares the device or the file for the ensuing data transfer. The .OPEN request has five forms; the desired form may be specified by inserting the proper HOW OPEN code in the File Block (see Figure 3-8) or by selecting one of the alternate forms of the Macro Call. The different .OPEN forms are described below:

Form	HOW OPEN Code	Description
.OPENU	1	Opens a previously created contiguous file for input and output by .RECRD or .BLOCK request; .OPENU is rejected if the device is not file-structured or if it is magtape or cassette.
.OPENO	2	a. Creates a new linked file and prepares it for output via .WRITE; the file must not already exist. b. Prepares a non-file-structured device for output via .WRITE (e.g., punch a leader for paper tape output).
.OPENE	3	Opens a previously created linked or contiguous file to make it longer via .WRITE; note that a contiguous file may only be extended within the area already allocated; although additional blocks may be added to a linked file, no additional blocks may be added to a contiguous file (see .CLOSE); .OPENE is treated like .OPENO if the device is not file-structured.
.OPENI	4	a. Opens a previously created linked or contiguous file for input via .READ, .RECRD, or .BLOCK. b. Prepares a non-file-structured device for input via .READ.
.OPENC	13	Opens a new contiguous file for output via .WRITE. When a contiguous file is first opened for writing (via .WRITE), .OPENC must be used. Subsequent opens for output (via .WRITE) must be .OPENE's. The .OPENC request is treated like .OPENO if the device is not file-structured. .OPENC is the only request which can create a contiguous file (via .WRITE).

At this point, the user should note the difference between linked files and contiguous files. A linked file has records allocated to it one at a time, as they are needed. Each record in the file contains a pointer to its successor, the User File Directory (UFD) points to the first record. Because records are allocated as needed, the user need not concern himself at all with the size of the file nor with the allocation of any records. Furthermore, a linked file can easily be extended in the future. However, because records are scattered about on the disk and

because the system must read all intermediate records to move from one record to another (forward only), linked files can only be used for sequential processing (.READ or .WRITE).

A contiguous file has all of its records allocated at once in a contiguous area of the disk which is reserved for the file. Since any record in the file can easily be located relative to the first record in the file, random (or direct) access (.RECRD or .BLOCK) is possible in addition to sequential access. However, it is now necessary to know in advance how much space will be needed, since no more space can be added later. Since this may be difficult, one often has to guess and space is often wasted. Note, however, that a contiguous file can be extended within the space already allocated, i.e., if the area was not filled when the file was first written (or extended), more data can be added. Because the user is responsible for determining the size of a contiguous file, he is required to allocate it before opening it (compare .OPENC and .OPENO). This may be done with PIP, using the ALLOCATE command or with the .ALLOC programmed request.

After the open request has been processed, control is returned to the user at the instruction following the assembly language expansion; the arguments are removed from the stack. At this time, however, the device concerned may still be completing operations required by the request. A summary of transfer requests which may legally follow .OPEN requests is illustrated in Table 3-6.

Table 3-6
Transfer Requests Which May Follow Open Requests

	Linked File		Contiguous File				
	Input	Output	Input		Output		File Already Exist?
Type of Open	.READ	.WRITE	.READ	.RECRD .BLOCK	.WRITE	.RECRD .BLOCK	
.OPENU				Yes		Yes	Must
.OPENO		Yes					Must Not
.OPENE		Yes			Yes		Must
.OPENI	Yes		Yes	Yes			Must
.OPENC					Yes		Must

Rules: a. General Rules for All .OPENx Requests - The user must set up a Filename Block in his program (see Figure 3-8). If the dataset is a file, the Filename Block must contain a legal filename. If the dataset is not a file, or if it will be specified by an .ASSIGN or via the Command String Interpreter, the Filename Block need not contain any filename or extension entries.

All datasets must have been INITed before they are OPENed. The .OPEN must be applicable to the type of device (e.g., .OPENI to the line printer is illegal).

For datasets on directory devices, the User Identification Code (UIC) in the Filename Block (if specified) must be in the directory of the device. If the UIC is not specified, the default UIC is that of the current logged in user.

The .OPENx request must not violate the protect code of the file.

If a dataset is opened for any output, it cannot be opened again until it has been closed.

b. Rules for .OPENO - The .OPENO request is applicable only for output to non-file-structured devices or to a linked file on a file-structured device. It is not applicable to contiguous files.

The .OPENO request creates a linked file on a file-structured device; hence, the file referenced in the corresponding Filename Block cannot exist prior to the .OPENO request.

The .OPENO request will return an error if the disk is full.

c. Rules for .OPENI - .OPENI may be used for inputs from contiguous or linked files, or nondirectory devices.

The file referenced in the corresponding Filename Block must exist in the directory.

If a file is open for input (.OPENI), it cannot be opened for output, but it may be opened for extension or update.

At any one time, a file can be opened for input to a maximum of 62_{10} or 76_{8} datasets.

d. Rules for .OPENU, OPENE, and .OPENC - The file must exist and cannot currently be opened for output.

The file cannot currently be opened by another .OPENU, .OPENE, or .OPENC.

A contiguous file can be opened for extension, provided that the area already allocated to the file does not need to be enlarged.

A linked file cannot be opened with .OPENC, which is applicable only to contiguous files.

Errors: If any of the preceding rules are violated, the Monitor places an error code in the STATUS byte of the Filename Block (see Table 3-7) and transfers control via the pointer in the ERROR RETURN ADDRESS of the Filename Block. If this address is Ø, a fatal error message is printed on the terminal. Fatal error messages are listed in Appendix K.

Example: See the .CLOSE request.

.O2BIN 3.6.26 .O2BIN - Convert six octal ASCII characters into one binary word.

Macro Call: .O2BIN #ADDR
where ADDR is the address of the first byte in the 6-byte string of octal characters to be converted.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #4,-(SP)              ;MOVE CALL CODE ONTO STACK
EMT 42
```

Global Name: CVT.

Description: The 6-byte string of 7- or 8-bit octal ASCII characters which starts at ADDR is converted into the binary number equivalent. The converted value is returned to the top of the stack, right-justified, followed by the address of the byte which follows the last character converted. The largest octal number which can be converted is 177777. The stack must be cleared by the user.

Errors: The conversion will be stopped if an error condition is encountered, and the user will be informed of the type of error via the condition codes in the Processor Status register:

C-bit set means that a byte was not an octal digit.

V-bit set means that the octal number was too large, i.e., the first byte was greater than 1.

If the conversion has been satisfactory, the condition codes are cleared. Following C- or V-bit errors, the value returned will be correct up to the last valid byte. The address returned will be that of the first invalid byte.

3.6.27 Requests to Perform Conversions

Using the EMT level 42 instruction the user can request data conversions between binary and some external form such as decimal ASCII or Radix-5Ø. He communicates his request by pushing the necessary parameters and an identifier code onto the stack. If a code outside the range of those currently established is specified, a fatal error (FØ34) will result.

.RADPK 3.6.27.1 .RADPK - Pack three ASCII characters into one Radix-5Ø word.

Macro Call: .RADPK #ADDR
where ADDR is the address of the first byte in the 3-byte string of ASCII characters to be converted.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
CLR -(SP)                    ;MOVE CALL CODE ONTO STACK
EMT 42
```

Global Name: CVT.

Description: The string of 7- or 8-bit ASCII characters in three consecutive bytes starting at ADDR is converted to Radix-5Ø packed ASCII using the algorithm shown below. The packed value is returned on the top of the stack, followed by the address of the byte following the last character converted. The user must clear the stack.

Radix-5Ø is used by the Monitor to store in one word three characters for half a filename or an extension or other three-character sets of data.

Because the characters allowed within names (e.g., filenames or extensions, assembler symbols, etc.) are restricted to letters, digits, and a few special characters, it is possible to store three characters within a single word by using the formula:

$$(((C_1 \times 50_8) + C_2) \times 50_8) + C_3$$

where C_1, C_2, and C_3 are the three characters converted from their original ASCII value to the value shown in the following table.

CHARACTER	ASCII Value	Radix-5Ø Value
Space	4Ø	Ø
A-Z	1Ø1-132	1-32
$	44	33
.	56	34
unused		35
Ø-9	6Ø-71	36-47

The maximum value for three characters is thus:

$$(((47 \times 50) + 47) \times 50) + 47 = 174777$$

The Radix-5Ø representation for various peripheral devices is shown below:

Mnemonic	Device	Radix-5Ø Equivalence
CR	Card Reader (CR11)	Ø1262Ø
DC	RC11 Disk	Ø1457Ø
DF	RF11 Disk	Ø1476Ø
DK(A,B)	RK11 Disk	Ø1527Ø(+1,2)
DT(A)	DECtape (TC11)	Ø16Ø4Ø(+1)
KB	ASR-33 Keyboard/Printer	Ø4242Ø
LP	Line Printer (LP11)	Ø466ØØ
MT	Magtape (TM11)	Ø5214Ø
PP	High-Speed Paper Tape Punch	Ø632ØØ
PR	High-Speed Paper Tape Reader	Ø6332Ø
PT	ASR-33 Paper Tape Device	Ø6344Ø
CT	TA11 Cassette Tape	Ø1274Ø

Rules: 1. Device mnemonics may be three letters on some systems. The third letter is assigned if there is more than one controller. For example:

DTA for DECtape controller A
DTB for DECtape controller B

2. The device name may be followed by an octal number to identify a particular unit when the controller has several device units associated with it. For example:

DT1 for unit 1 under a single DECtape control
DTA1 for unit 1 under controller A in a multi-controller situation.

Errors: The conversion will be stopped if an error condition is encountered, and the user will be informed of the type of error via the condition codes in the Processor Status register:

C-bit set means that an ASCII byte outside the valid Radix-5Ø set was encountered.

The value returned will be left-justified and correct up to the last valid byte, e.g., DT: will be returned packed as DTΔ. The address returned will be that of the first invalid byte.

If no errors were encountered during the conversion, the condition codes will be cleared.

Example: Pack a string of $3Ø_{1Ø}$ ASCII characters, starting at UNPBUF, into a buffer starting at PAKBUF.

```
        MOV #PAKBUF,R3          ;SET UP POINTER TO PACK BUFFER
        MOV #UNPBUF,-(SP)       ;.RADPK UNBUF
NEXT:   CLR -(SP)
        EMT 42
        BCS ERRC                ;INVALID ASCII CODE ENCOUNTERED
        MOV (SP)+,(R3)+         ;MOV PACKED VALUE TO BUFFER
        CMP R3,#PAKBUF+12       ;END OF STRING?
        BNE NEXT                ;NO
        TST (SP)+               ;YES - REMOVE POINTER FROM STACK
```

Note that this example takes advantage of the fact that the Monitor returns on the stack the address of the byte which follows the last character converted.

3.6.27.2 .RADUP - Unpack one Radix-5Ø word into three ASCII characters. .RADUP

Macro Call: .RADUP #ADDR,#WORD

where ADDR is the address of the first of three bytes into which the unpacked characters are to be placed, and WORD is the Radix-5Ø word to be converted.

Assembly Language Expansion:

```
        MOV #WORD,-(SP)
        MOV #ADDR,-(SP)
        MOV #1,-(SP)            ;MOVE CALL CODE ONTO STACK
        EMT 42
```

Global Name: CVT.

Description: WORD is converted into a string of 7-bit ASCII characters which are placed left-justified with trailing spaces in three consecutive bytes starting at location ADDR. The stack is cleared. See Section 3-3.6.27.1 for a definition of Radix-5Ø.

Errors: If an error is encountered, the user will be informed via the condition codes in the Processor Status register.

C-bit set means:

a. a value of WORD was outside the valid Radix-5Ø set, i.e., >174777 (see Section 3-3.6.27.1).

b. a Radix-5Ø byte value was found to be 35, which is currently not used.

Nevertheless, three bytes will be returned with a : as the first of the three for error type (a), and a / for any of the three bytes for error type (b).

If the conversion is satisfactory, the condition codes are cleared.

.READ 3.6.28 .READ - Read the next record in the dataset.

Macro Call: .READ #LNKBLK,#BUFHDR
where LNKBLK is the address of the Link Block, and BUFHDR is the address of the line buffer header.

Assembly Language Expansion:

```
MOV #BUFHDR,-(SP)
MOV #LNKBLK,-(SP)
EMT 4
```

Global Name: RWN. (Routine is permanently core resident.)

Description: The .READ request transfers the data from the device to the user's line buffer as specified in the line buffer header. The transfer is done via a buffer in the Monitor, into which an entire device block is read, and from which the desired data is transferred to the user's line buffer. Each read causes the user to receive the next record in the data set. Block boundaries are ignored and new blocks are read as needed. After any I/O transfer has been started, control is returned to the user at the next instruction, with the arguments removed from the stack.

Refer to Section 3-4.3.1 for more details on transfer modes.

Rules: If the device is file-structured, the .READ request must be preceded by an .OPENI. The user must provide in his program a line buffer and line buffer header (see Figure 3-9). Further actions on the dataset by the Monitor will be automatically postponed until the .READ processing has completed. The user program should, however, perform a .WAIT or .WAITR to ensure proper completion of transfer before attempting to use the data in the line buffer. Otherwise, the program might find that it is processing before the data it wants has arrived.

Errors: Specification of a transfer mode which is inappropriate for the device assigned to the dataset, attempting to .READ from or .WRITE to a file-structured device for which no file has been .OPENed or for which the type of .OPEN is incorrect, will be treated as fatal errors and will result in a FØ1Ø message.

Note: A dataset can only support transfers in one direction at one time, i.e., READ only or WRITE only. If the same device is to be used for both operations, separate datasets must be used for each.

3.6.29 .RECRD - Read or write a specific record in a file.

Macro Call: .RECRD #LNKBLK,#RECBLK
where LNKBLK is the address of the Link Block, and RECBLK is the address of the Record Block (see Figure 3-12).

Assembly Language Expansion:

```
MOV #RECBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 25
```

Global Name: REC.

Description: The .RECRD request causes a specific record to be transferred to (or from) the user's record buffer. Each record in the file may be individually addressed, and the user is not restricted to reading or writing the next record. Data transfer is by way of a buffer in the Monitor which contains exactly one physical block of information. There is no rule concerning the relative sizes of records and blocks; however, efficiency may be improved if either is a multiple of the other. The Record Block specifies record number (starting at Ø), buffer address and length, and transfer direction (read or write). .RECRD requests require the use of the .INIT, .RLSE, .OPEN, .CLOSE, and .WAIT (or .WAITR) requests. After the transfer has started, control is returned to the user at the instruction following the assembly language expansion with arguments removed from the stack. This function is not supported for magtape or cassette.

Rules:
1. The requested device must be file-structured and the file must be contiguous.
2. The user must set up a Record Block in his program and must provide a buffer.
3. All records must have the same length.
4. The user should perform a .WAIT or .WAITR to ensure that processing has completed.
5. The associated file must have been opened with .OPENU or .OPENI.

Errors: An error causes a return to the user with the type of error indicated in the FUNCTION/STATUS word of the RECORD Block. The user should perform the following test after his request to ensure that the request completed normally.

```
TSTB  RECBLK+1
BNE   ERROR
```

.RENAM 3.6.30 .RENAM - Rename a file. Change protection code.

Macro Call: .RENAM #LNKBLK,#OLDNAM,#NEWNAM

where LNKBLK is the address of the Link Block, OLDNAM is the address of the Filename Block representing the file, and NEWNAM is the address of the Filename Block containing the new information.

Assembly Language Expansion:

```
MOV #NEWNAM,-(SP)
MOV #OLDNAM,-(SP)
MOV #LNKBLK,-(SP)
EMT 2Ø
```

Global Name: REN. (See Chapter 3-5 for subsidiary routines.)

Description: Allows the user to change the name and protection code (see Section 3-2.4) of a file. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack.

Rules: Dataset must be INITed, and file must not be OPENed. The user must specify two Filename Blocks: one contains the name and protection code of the file as it presently is before the .RENAM request, and the other contains the name and protection code of the file as it should be after the .RENAM request. The two filenames must be different. To change just the protection for a file, two .RENAMs must be requested.

The new filename must not already exist, and the new filename must be legal. The old file must exist. Renaming a file assigned to the keyboard will effectively be a NOP.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the offending Filename Block if it is specified and applicable, or to the Monitor for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	--	FØØØ
File Exists (new name)	2	FØ24
File Nonexistent (old file)	2	FØ24
Protection Violation	6	FØ24
File Is Open	14	FØ24
Illegal Filename	15	FØ24

3.6.31 .RLSE - Remove the linkage between a device driver and a dataset and release the driver. **.RLSE**

Macro Call: .RLSE #LNKBLK
where LNKBLK is the address of the Link Block previously INITed.

Assembly Language Expansion:

```
MOV #LNKBLK,-(SP)
EMT 7
```

Global Name: RLS.

Description: Dissociates the device from a dataset and releases the dataset's claim to the driver. Releasing the driver frees core provided no other dataset has claimed the driver, and provided that the driver is not permanently core resident.

Rules: The device to be released must have been previously INITed to the dataset.

If the dataset has been OPENed on a directory device, it must be CLOSEd before the device is released. On a nondirectory device, or on magtape and cassette, a .RLSE will ensure that any data remaining in the Monitor buffer for output is dispatched to the device and will return any buffer still associated with the dataset to free core.

After the release has been completed, control is returned to the user at the instruction following the assembly language expansion; the argument is removed from the stack.

Errors: If the dataset has been OPENed to a file-structured device, a .RLSE not preceded by a .CLOSE will be treated as a fatal error, FØØ5. A .RLSE error (FØØ5) may also occur if the link pointer in the Link Block is invalid, indicating probable corruption of the Monitor or its control blocks.

Example:

```
            .
            .
            .
        .INIT   #LNK1       ;ASSOCIATE A DATASET WITH A DEVICE
            .
            .
            .
        .RLSE   #LNK1
            .
            .
            .
        .WORD   ERR1        ;ERROR RETURN ADDRESS
LNK1:   .WORD   Ø            ;POINTER FOR MONITOR
        .RAD5Ø  /DSI/       ;LOGICAL NAME OF DATASET
        .BYTE   1,Ø         ;DEVICE SPECIFIED, UNIT
        .RAD5Ø  /KB/        ;SPECIFY KEYBOARD
            .
            .
            .
ERR1:                       ;ERROR PROCESSING LOGIC
```

.RSTRT 3.6.32 .RSTRT - Set the default address for use by the REstart keyboard command.

Macro Call: .RSTRT #ADDR

where ADDR is the restart address.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #2,-(SP)        ;2 is the identifier code for .RSTRT
EMT 41
```

Global Name: GUT.

Description: Sets the address where the program should restart in response to the keyboard command REstart. This is the assumed address in the absence of an address in the REstart command. It can be reset as often as requested by the program. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared.

Rules: ADDR must be an address within the user's core area.

.RUN 3.6.33 .RUN Load and process the program.

Macro Call: .RUN #RUNBLK

where RUNBLK is the address of the user's Run Block (see Figure 3-15).

Assembly Language Expansion:

```
MOV #RUNBLK,-(SP)        ;PUSH ADDRESS OF THE RUN BLOCK
EMT 65                   ;ONTO THE STACK
```

Global Name: RUN.

Description: The RUN request may be used to load an entire program or a program overlay.

1. Load a program or load an overlay - when an overlay is loaded, the existing program environment is not disturbed; one section of the program is simply replaced by another. When a new program is loaded, the old program and its effects (except for data on the stack) are purged from core, and the new program takes over; for example, FORTRAN can use the RUN request to load LINK and LINK can use it to load and execute the user's program;

2. Load a core image or a load module;

3. Return of control:

   instruction following .RUN;

   transfer address of load module or core image;

   transfer address plus offset (word F);

   alternate return address (word G);

4. Stack movement:

   leave as is;

   move the stack down if it would otherwise be destroyed by the entity being loaded; (stack movement does not occur on short form calls).

5. Load address:

   as specified in file,

   as specified by user.

The RUN request requires the following control blocks:

Run Block: A variable length control block whose address is passed on the stack. It contains a function word and various optional parameters. It is described in Section 3-4.7.

Link Block: The standard Link Block (Section 3-4.1). It describes the device from which the entity is to be loaded. It is required unless bit 15 of the function word in the Run Block is 1.

File Block: The standard File Block (Section 3-4.2). It describes the file from which the entity is to be loaded: either an .LDA file or a CIL. It is required unless bit 15 of the function word in the Run Block is 1.

The Link Block should not be .INITed, nor should the File Block be .OPENed, when .RUN is called. RUN will perform .OPEN, .CLOSE, .INIT and .RLSE processing. The lookup sequence is as follows:

First an extension of LDA is attempted, then no extension, unless an extension is specified, in which case it alone is used;

For each extension, the current UIC, then [1,1] is tried, unless a UIC is specified, in which case it alone is used.

The .RUN request always removes the Run Block address from the stack. If bit Ø is Ø, the following information will be returned upon the stack:

(SP) - transfer address of loaded module,
2(SP) - size of loaded module in bytes,
4(SP) - low address of loaded module.

Aside from this, the stack is not disturbed, although it may be moved. This means that the stack may be used for passing arguments.

Rules: 1. The Link Block should not be .INITed.

2. The File Block should not be .OPENed.

3. If an overlay is being loaded, it must not extend above the bottom of the resident program section, nor below the top of the Monitor.

4. If a new program is to be loaded, all datasets used by the current program must be RLSEd.

5. The user must be sure that his stack is not inadvertently destroyed.

6. The appropriate supporting data must be present in the RUN Block for the options are requested through the function word.

7. If the stack might be moved, it must not contain absolute pointers to locations within the stack. For example:

```
MOV  SP,RØ
MOV  RØ,-(SP)
```

produces a stack which should not be moved. The user can assure that such a stack will not be moved by setting bit 1 of the Function word in the RUN Block to Ø (see Section 3-4.7).

Errors: Errors FØØ7, FØ12, FØ21, FØ22, FØ24, FØ45, FØ54, F274, F276, and F277 are all possible. All but FØØ7 and FØ21 are nonfatal, provided that an error return is provided in the File Block (see Table 3-8).

.SPEC 3.6.34 .SPEC - Special functions.

Macro Call: .SPEC #LNKBLK,#SPCARG

where LNKBLK is the address of the Link Block, and SPCARG may be either a special function code or the address of a special function block containing the code (see Figure 3-17), depending upon the function.

Assembly Language Expansion:

```
MOV #SPCARG,-(SP)
MOV #LNKBLK,-(SP)
EMT 12
```

Global Name: SPC.

Description: This request is used to specify a special function (action) to a device, such as rewind magnetic tape. A code identifies the function and must be in the range Ø-$255_{1Ø}$. When the function requires no supporting data, the code itself is the first parameter to be placed upon the processor stack in the assembly

language call sequence. However, if the user must supply additional information or if the function expects to return data to the user, the code is passed within a special function block and the address of the block is the call parameter. The format of this block is illustrated in Figure 3-17, Section 3-7.1.1.

If a .SPEC request is made to a device which has no special function code, an immediate return is made showing that the function has been complete. After the request has been started, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared.

For further information, see Appendix C, "Physical Device Names", and Chapter 3-7 for special functions.

Rules: The dataset must be INITed.

Errors: Fatal error FØØØ is returned if the dataset has not been INITed.

3.6.35 .STAT - Obtain device status.

.STAT

Macro Call: .STAT #LNKBLK
where LNKBLK is the address of the Link Block.

Assembly Language Expansion:

```
MOV #LNKBLK,-(SP)
EMT 13
```

Global Name: STT.

Description: Determine for the user the characteristics of the device specified in the Link Block. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. This request returns to the user with the following information at the top of the stack.

SP	Driver Facilities Word
SP+2	Device Name (Packed Radix-5Ø)
SP+4	Device Standard Buffer Size (in words)

The Driver Facilities Word has the following format.

Table 3-7
Driver Facilities Word Format

Bit	Switch = 1 (ON) indicates
Ø	Device will support multidataset activity.
1	Device will handle output.
2	Device will handle input.
3	Device will handle binary data.
4	Device will handle ASCII data.
5	Driver has a special function entry.
6	Driver has a CLOSE entry.
7	Driver has an OPEN entry.
8	Device is a terminal.
9	Device is a sequential cassette tape.
1Ø	Device has multiple units under one Controller.
11	Device supports multiple record lengths.
12	Device is the system disk driver.
13	Device is sequential magnetic tape.
14	Device is DECtape.
15	Device is directory (file) structured.

Device Name is the Radix-5Ø packed ASCII standard mnemonic for the device (Appendix C); and, Device Standard Buffer Size is the block size (in words) on a blocked device or an appropriate grouping size on a character device.

Rules: The dataset must be INITed. The user must clear the stack upon return.

.STFPU 3.6.36 .STFPU - Initialize the floating-point exception vector.

Macro Call: .STFPU #PSW,#ADDR

Assembly Language Expansion:

```
MOV #ADDR,-(SP)      ;ADDRESS OF EXCEPTION ROUTINE
MOV #PSW,-(SP)       ;PROGRAM STATUS WORD FOR
                     ;EXCEPTION RTN
MOV #3,-(SP)         ;REQUEST CODE
EMT 41
```

Global Name: GUT.

Description: This request initializes the exception interrupt vector for the floating-point processor on the PDP-11/45 or the FIS instruction on the PDP-11/4Ø.

Any floating-point exception for which interrupt is enabled will cause a trap to location ADDR with a new program status word of PSW. The interrupt vector is at location 244_8.

3.6.37 .STPLA - Set the program low address. .STPLA

Macro Call: .STPLA #ADDR
where ADDR is the desired new program low address.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #5,-(SP)
EMT 41
```

Global Name: GUT.

Description: This request allows the user to establish a new program low address. This is done if the user wants part of his resident code overlayed or if he wants to reserve additional space between his resident code and his overlays. Consult the .GTPLA description for more details.

The old program low address (or a zero) will be returned on top of the stack upon return from this macro call. The stack is not moved.

Rules: The user is required to clear the returned address from the stack.

Errors: The address returned on top of the stack will be zero when the call is unsuccessful. This occurs when the address is outside of available memory.

3.6.38 .STSTK - Modify the stack base entry. .STSTK

Macro Call: .STSTK #ADDR
where ADDR is the desired new stack base address entry.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #4,-(SP)
EMT 41
```

Global Name: GUT.

Description: This request is used when the stack is to be relocated. It does not relocate the stack, but it does record its new base (the address of the word immediately above the stack; see Section 3-3.6.19), and it returns the old stack base on the stack. EXTREME CAUTION should be used when moving the stack; it is

not recommended as a standard procedure. Note that the .RUN request may be used to move the stack when that is appropriate.

Rules: The user must clear the old base value from the stack when control is returned.

The user is responsible for moving the stack.

Caution should be used when moving the stack, since the new and old stack areas may overlap and since Monitor interrupt routines may use the stack while it is being moved. Let:

```
SB1 = old stack base (returned on stack)
SB2 = new stack base (supplied by user)
SP1 = old stack pointer (current value of SP)
SP2 = new stack pointer (SB2 - SB1 + SP1)
```

First, set SP=min (SP1,SP2) to protect against interrupts. Then if SB1<SB2, move the stack starting from the base (SB1 to SB2). If SB1>SB2, move the stack starting from the top (SP1 to SP2). This strategy prevents the stack from being corrupted during the move (since the two stack areas might overlap). Finally, set SP to SP2.

Errors: If the new stack base ADDR is outside available memory or inside the Monitor, the request is not honored and a zero is returned on the stack.

.SYSDV 3.6.39 .SYSDV - Get name of the system device.

Macro Call: .SYSDV

Assembly Language Expansion:

```
MOV #1Ø6,-(SP)
EMT 41
```

Global Name: GUT.

Description: The name of the system device in Radix-5Ø notation is returned to the user at the top of the stack.

Rules: The user must clear the stack.

3.6.40 .TIME - Obtain current time of day. **.TIME**

Macro Call: .TIME

Assembly Language Expansion:

```
MOV #1Ø4,-(SP)
EMT 41
```

Global Name: GUT.

Description: The two current time words are returned to the user at the top of the stack.

SP:	LOW-ORDER TIME IN TICKS
SP+2:	HIGH-ORDER TIME

where a TICK is 1/6Ø of a second (1/5Ø second for 5Ø-cycle lines).

The words are 15-bit unsigned numbers. See the CVTDT request for how to obtain the ASCII representation of current time value.

Rules: The user must clear the stack.

3.6.41 .TRAN - Read or write the specified block (file-structured device) or the next block (non-file-structured device). **.TRAN**

Macro Call: .TRAN #LNKBLK,#TRNBLK
where LNKBLK is the address of the Link Block, and TRNBLK is the address of the TRAN block (see Figure 3-14, Section 3-4.6).

Assembly Language Expansion:

```
MOV #TRNBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 1Ø
```

Global Name: TRA.

Description: .TRAN provides nearly direct access to the device on which the dataset resides. No file processing is done and any file structure is ignored. Writing with .TRAN on a file-structured device is especially risky and can lead to the corruption of all data on the device. If .BLOCK request can be used instead of .TRAN, it is recommended. Each .TRAN will transfer one or more blocks, depending upon the WORD COUNT in the TRAN Block. Blocks on file-structured devices are referenced by absolute block number, while blocks on non-file-structured

devices are processed in sequence. .INIT, .RLSE and .WAIT (or .WAITR) must be used, while .OPEN and .CLOSE must not. After the transfer has started, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack.

Rules: .TRAN must be preceded by an .INIT request on the associated dataset. .OPEN must not be used. For each .TRAN request, the user must provide a transfer control block, as shown in Figure 3-14. Further actions on the dataset by the Monitor will be automatically postponed until the .TRAN processing has been completed. The user program should perform a .WAIT or .WAITR to ensure proper completion of the transfer before attempting to reference any location in the data buffer.

Errors: An invalid function code in the transfer control block will result in an error diagnostic message on the terminal at run time.

Errors in the transfer will be shown in the FUNCTION/STATUS word of the TRAN block; the last word of the block will be set to show how many data words have not been transferred.

Example: Transfer 200_8 words of data from DECtape unit 3, starting at block 100_8 to core starting at location BUFFER.

```
            .INIT #TAPE1              ;INITIATE DATASET
              :
              :
            .TRAN #TAPE1, #BIN40      ;INITIATE TRANSFER
              :
              :
            .RLSE #TAPE1              ;RELEASE DATASET
              :
              :
            .WORD ERR1                ;LINK BLOCK
TAPE1:      .WORD 0
            .RAD50 /TP1/
            .BYTE 1,3
            .RAD50 /DT/
              :
              :
BIN40:      .WORD 100                 ;STARTING BLOCK #
            .WORD BUFFER              ;STARTING ADDRESS IN CORE
            .WORD 200                 ;NUMBER OF WORDS
            .WORD 4                   ;INPUT
            .WORD 0                   ;FOR MONITOR USE
              :
              :
ERR1:                                 ;ERROR ROUTINE FOR DECTAPE
              .
              :
              :
BUFFER:     .BLKW 200
BUFEND:     .WORD 0
              :
              :
            .END
```

3.6.42 .TRAP - Set interrupt vector for the trap instruction. .TRAP

Macro Call: .TRAP #STATUS,#ADDR
where STATUS is the desired status for the trap, and ADDR is the address for the trap.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #STATUS,-(SP)
MOV #1,-(SP)              ;1 is the identifier code for .TRAP
EMT 41
```

Global Name: GUT.

Description: Sets the STATUS and ADDR into trap vector 34. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared. The user may then use the trap instruction.

Rules: STATUS must be a valid Status Byte.
ADDR must specify an address within the user's core area.

Errors: If an invalid code is specified, a fatal (FØØ2) error will result.

3.6.43 .WAIT - Wait for completion of process on dataset. .WAIT

Macro Call: .WAIT #LNKBLK
where LNKBLK is the address of the Link Block (see Figure 3-7).

Assembly Language Expansion:

```
MOV #LNKBLK,-(SP)
EMT 1
```

Global Name: None. (Routine is embedded in the resident Monitor.)

Description: .WAIT tests for completion of the last requested action on the dataset represented by the referenced Link Block. If the action is complete (that is, if the request has completed all its action), control is returned to the user at the next sequential instruction following the assembly language expansion; otherwise, the Monitor retains control until the action is complete. A .WAIT or .WAITR should be used to ensure the integrity of data transferred to or from a line buffer. The argument is removed from the stack.

Rules: The dataset must be INITed.

Errors: If the dataset is not INITed, a fatal error occurs and FØØØ is printed on the terminal.

.WAITR 3.6.44 .WAITR - Check for completion of processing on dataset and return or transfer.

Macro Call: .WAITR #LNKBLK,#ADDR
where LNKBLK is the address of the Link Block, and ADDR is the address to which control is transferred if the processing is not complete.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #LNKBLK,-(SP)
EMT Ø
```

Global Name: (Routine is imbedded in the resident Monitor.)

Description: .WAITR tests for completion of the last requested action on the specified dataset. If all actions are complete, control is returned to the user at the next sequential instruction following the assembly language expansion. If all actions are not complete, control is given to the instruction at location ADDR. The arguments are removed from the stack. It is the user's responsibility to return to the .WAITR to check again.

Rules: The user should use a .WAIT or a .WAITR request to assure the completion of data transfer to the user's line buffer before processing the data in the buffer, or moving data into it. The dataset must be INITed.

Errors: If the dataset is not INITed, a fatal error occurs and FØØØ is printed on the terminal.

.WRITE 3.6.45 .WRITE - Write the next record in the dataset.

Macro Call: .WRITE #LNKBLK,#BUFHDR
where LNKBLK is the address of the Link Block, and BUFHDR is the address of the line buffer header.

Assembly Language Expansion:

```
MOV #BUFHDR,-(SP)
MOV #LNKBLK,-(SP)
EMT 2
```

Global Name: RWN. (Routine is permanently core resident.)

Description: The .WRITE request initiates the transfer of data from the user's line buffer to the device assigned. The data is first transferred to a buffer in the Monitor, where it is accumulated until a buffer of suitable length for the device is filled.[1] The data in the Monitor buffer is then transferred to the next device block, and any data remaining in the user's line buffer is moved to the (now emptied) Monitor buffer. After any I/O transfer to the device has been started, control is returned to the user at the next sequential instruction. The arguments are removed from the stack upon return.

Refer to Section 3-4.3.1 for more details on transfer modes.

Rules: If the requested device is file-structured, the dataset must have been opened by an .OPENO or .OPENE for a linked file, or .OPENC for a contiguous file. The user must provide a line buffer and its header in his program (Figure 3-9).

Further actions on the dataset by the Monitor after .WRITE will be automatically postponed until the .WRITE processing has been completed. Before refilling the line buffer, however, the user program should perform a .WAIT or .WAITR to ensure proper completion of the transfer. Otherwise, it might store new data on top of data which has not yet been written.

Errors: See .READ for errors.

3.6.46 .DUMP - Dump specified core locations to the line printer. This Monitor directive can only be used in Batch mode. **.DUMP**

Macro Call: .DUMP LOWADD,HIADD,CODE
where LOWADD contains the lower bound of core to be dumped, HIADD contains the upper bound, and CODE is zero.

Assembly Language Expansion:

```
MOV LOWADD,-(SP)
MOV HIADD,-(SP)
MOV CODE,-(SP)
EMT 64
```

Global Name: DMP.

Description: A user running in Batch mode can request "dump-on-error" with the /DU switch on the $RUN or $GET command. When an error occurs, EMT 64 is called to dump

[1]For terminal devices, data transfer also occurs when a line terminator is seen (see Section 3-4.3.2).

core to a system disk file, DMP.SYS, and to abort the job. When control is returned to the Monitor, it runs a program in the file LDUMP.LDA as part of its job termination processing. LDUMP formats and prints the contents of DMP.SYS. See System Managers Guide "Job Termination Processing" if no line printer is available. EMT 64 can also be invoked by a user program via .DUMP.

The register preamble is always dumped. With low and high address the user can specify dump limits, or he can use defaults:

A. High and low limits of Ø result in dump of program area.

B. A high limit of -1 defaults to the top of core.

C. Low and high limits of +1 result in a dump of the preamble only.

Note that in all cases the user must be running in batch mode and must include the /DU switch in the $RUN directive. The argument on top of the stack should always be Ø.

In addition, it is possible for the user to write his own dump formatting program to be run at job termination instead of LDUMP. (See System Manager's Guide, Section 5-3, on Job Termination Processing.)

EMT 64 writes unformatted binary data to DMP.SYS. The format follows:

Word 1	Flag Word
Word 2	Starting address of dump
Word 3	Number of bytes dumped
Word 4	SP
Word 5-12	RØ-R5
Word 13	PC
Word 14	PS
Words 15-17	Three arguments with which DUMP was called (Ø, high address, low address)
Words 2Ø...	Dump of number of bytes specified in Word 3.

The flag word's values are:

Ø2	User-specified limits
Ø4	Default high core value
Ø6	Default to program area
1Ø	Preamble only

Rules: 1. The system must be in Batch mode.

2. The DMP.SYS file must be unlocked and present in UIC [1,1].

3. The /DU switch must appear with the RUN or GET command.

Errors: If DMP.SYS is locked, the Monitor directive will be flagged with the 'ILL CMD' message.

The command is ignored if the /DU switch was not on the RUN or GET command or if the system is in interactive mode.

3.6.47 .FLUSH - Bypasses lines in the batch stream. This Monitor directive can only be used in Batch mode. **.FLUSH**

Macro Call: .FLUSH CODE
where CODE contains the value Ø, 1 or 2.

Assembly Language Expansion:

```
MOV CODE,-(SP)
EMT 67
```

Global Name: BSF. (Batch Stream Flush)

Description: .FLUSH allows the currently running program (system or user) to request that batch stream input be bypassed until a specified type of control card is encountered. The type of control card is specified by the contents of code.

CODE	STATEMENT TYPE
Ø	$
1	$ or #
2	$, #, or *

.FLUSH is a NOP when the system is not in Batch mode.

Rules: The system must be in Batch mode.

Errors: If an illegal .FLUSH code is input, a fatal error occurs and FØ53 is printed on the terminal.

3.7 EMT CODE SUMMARY

EMT Code	Programmed Request	Described on Page
Ø	.WAITR	3-94
1	.WAIT	3-93
2	.WRITE	3-94
3	[2]	
4	.READ	3-80
5	[2]	
6	.INIT	3-68
7	.RLSE	3-83
1Ø	.TRAN	3-91
11	.BLOCK	3-54
12	.SPEC	3-86
13	.STAT	3-87
14	.LOOK	3-69
15	.ALLOC	3-50
16	.OPENx	3-72
17	.CLOSE	3-55
2Ø	.RENAM	3-82
21	.DELET	3-63
22	.APPND	3-52
25	.RECRD	3-81
26-27	[2]	
3Ø-31	[1]	
32	Diagnostic Print	
33-35	[1]	
36-37	[2]	
4Ø	[1]	
41	General Utilities	
	.CORE	3-56
	.DATE	3-62
	.GTCIL	3-65
	.GTCLK	3-65
	.GTOVF	3-66
	.GTPLA	3-66
	.GTRDV	3-67
	.GTSTK	3-67
	.GTUIC	6-68
	.MONF	3-71
	.MONR	3-72
	.RSTRT	3-84
	.STFPU	3-88
	.STPLA	3-89
	.STSTK	3-89
	.SYSDV	3-90
	.TIME	3-91
	.TRAP	3-93
42	General Conversions	
	.BIN2D	3-53
	.BIN2O	3-53
	.D2BIN	3-64
	.O2BIN	3-76
	.RADPK	3-76
	.RADUP	3-79

[1] Reserved for Monitor internal communication.
[2] Reserved for future Monitor expansion.

EMT Code	Programmed Request	Described on Page
43-55	[1]	
56,57	Command String Interpreter	
	.CSI1	3-57
	.CSI2	3-58
6Ø	.EXIT	3-64
61-62	[1]	
63	Magnetic Tape Open (MTO)	
64	[1]	
65	.RUN	3-84
66	.CVTDT	3-61
67	[2]	
68-69	[2]	
7Ø	[2]	
71	Cassette Tape Open (CTO)	
72-76	[2]	
77	[1]	
1ØØ-117	(reserved for Communications Executive, COMTEX-11)	
12Ø-137	(reserved for Real-Time Monitor, RSX-11)	
14Ø-167	(reserved for user-implemented routines)	

[1] Reserved for Monitor internal communication.
[2] Reserved for future Monitor expansion.

PART 3

CHAPTER 4

USER PROGRAM TABLES AND CONTROL BLOCKS

LINK Block

4.1 THE LINK BLOCK (used for all input/output and directory requests)

LNKBLK:

ERROR RETURN ADDRESS	
ØØØØØØ LINK POINTER (for Monitor use only)	
LOGICAL NAME OF DATASET -- Radix-5Ø Packed ASCII	
UNIT NUMBER	NUMBER OF WORDS TO FOLLOW
PHYSICAL DEVICE NAME -- Radix-5Ø Packed ASCII	

Figure 3-7
The Link Block

Each dataset in a user's program must have a Link Block associated with it. Entries in the Link Block which must be specified by the user can be written into his program or set by the program itself before the dataset is INITed. Each entry is explained below.

Address	Name	Function
LNKBLK-2	ERROR RETURN ADDRESS	This entry must be set by the user to contain the address where he wants to transfer control in the event that any request associated with this dataset fails to obtain required buffer space from the Monitor. If no address is specified here, such an error will be treated as fatal. This address may be changed by the user's program at any time.
LNKBLK	LINK POINTER	This location must be set to zero by the user and must not be modified by him. The Monitor places a linking address here when the dataset is INITed. Before INITing a dataset, the Monitor tests this pointer for zero. If it is not zero, the Monitor assumes that the dataset was already INITed.
LNKBLK+2	LOGICAL NAME OF DATASET	The user can specify a name for the dataset in this entry. This name, which must be unique, is used to associate the dataset with a device which is specified by an ASSIGN from the keyboard. The name is stored in Radix-5Ø packed ASCII by the .RAD5Ø assembler directive. This specification is optional, but if it is omitted, the ASSIGN command cannot be used.

Address	Name	Function
LNKBLK+4	NUMBER OF WORDS TO FOLLOW	This byte contains the count of the number of words to follow in the Link Block. The user should set it to a Ø if he does not specify any PHYSICAL DEVICE NAME in the next word, or to a 1 if he does. Values greater than 1 may be used if the Command String Interpreter is to be called (see Optional Data below).
LNKBLK+5	UNIT NUMBER	This code specifies the unit number of the device linked to the dataset. For example, the TC11 Controller (DECtape) can drive up to eight tape drives (units), numbered Ø-7.
LNKBLK+6	PHYSICAL DEVICE NAME	If the user specified 1 or greater in byte LNKBLK+4, he may specify here the standard name (Appendix C) for the device associated with the dataset in Radix-5Ø format. If no name is specified here, the user must specify LOGICAL NAME OF DATASET and perform an ASsign command before he runs his program. If physical device name is specified both here and in an ASSIGN command, the device specified in the ASSIGN command overrides the value given here.
LNKBLK+8 through LNKBLK+n	OPTIONAL DATA	Present only if LNKBLK+4 is greater than 1. It is used to pass additional information such as switch information when using the Command String Interpreter or Resident EMT information when using .RUN, via the Link Block

4.2 THE FILENAME BLOCK

FILENAME Block

Each file associated with a dataset must be described by the user in a Filename Block. If a dataset is not a file, the Filename Block must still be used (if .OPEN is used) but FILENAME, EXTENSION, and PROTECT need not be specified. The Filename Block is used by OPEN and all directory management requests.

FILBLK:

ERROR RETURN ADDRESS	
ERROR CODE	HOW OPEN
FILE NAME	
FILE NAME	
EXTENSION	
USER ID CODE	
(spare)	PROTECT CODE

Figure 3-8
The Filename Block

Address	Name	Function
FILBLK-4	ERROR RETURN ADDRESS	The user must specify here the address to which he wants the Monitor to return control if one of the errors in Table 3-8 occurs during an operation involving the file. If no address is specified here, any such error will be treated as a fatal error.
FILBLK-2	HOW OPEN	This is set when the .OPENx macro's assembly language expansion is executed. It tells the Monitor which kind of open is being requested: .OPENU=1, .OPENO=2, .OPENE=3, .OPENI=4, .OPENC=13.
FILBLK-1	ERROR CODE	This entry should not be set by the user. It will be set by the Monitor to indicate the type of error (Table 3-8) which occurred. It will be cleared of any previous condition at each .OPEN call.
FILBLK+Ø FILBLK+2	FILE NAME	This two-word entry must be specified by the user if this dataset, or a portion thereof, is a file. It is the name of the file, in packed Radix-5Ø format.
FILBLK+4	EXTENSION	This entry must be specified if the file named in the previous entry has an extension. It is in packed Radix-5Ø format.
FILBLK+6	USER ID CODE	The user may enter his USER ID CODE here in octal: GROUP NUMBER | USER'S NUMBER High-Order Byte | Low-Order Byte If no entry is specified here, the current user's UIC is assumed.
FILBLK+1Ø	PROTECT CODE	The user may specify here the protection to be given to the file at its creation or renaming. If Ø, a default protection 233 will be allotted.

Table 3-8
Filename Block Error Conditions

Error Code In File-name Block	Request Type	Cause	Remedy
ØØ	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to open a dataset that was previously opened.	
Ø1		unused	

(continued on next page)

Table 3-8 (cont.)
Filename Block Error Conditions

Error Code In File-name Block	Request Type	Cause	Remedy
Ø2	.OPENO	An attempt was made to open a file which already exists.	If name of file was correct, delete the file (with PIP) or change file name.
	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file for input, extension, or update which is currently opened for output, or which does not exist.	
	.RUN	The file specified was already OPENed for output, or the file does not exist.	
Ø3	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file which has already been opened the maximum number of times (76_8)	Close file.
Ø4	.OPENC .OPENE .OPENU	An .OPENC, .OPENE, or .OPENU attempt was made to open a file which has already been opened for either .OPENC, .OPENE, or .OPENU.	.CLOSE the previous open.
Ø5	.OPENE	Illegal request to a contiguous file.	
Ø6	.OPENC .OPENE .OPENI .OPENO .OPENU .RUN	An attempt was made to access a file which the protection code prohibits.	Resolve access problem with owner of the file.
Ø7	.OPENC	Illegal OPEN request to a contiguous file.	
11	.OPENC .OPENE .OPENO .OPENU	File opened for output or extension is already on current DECtape unit.	Close offending file.
12	.ALLOC .OPENO	Directory full (DT).	Mount another DECtape.
13	.ALLOC	The UIC was not entered into the device MFD.	Enter UIC via PIP.

(continued on next page)

Table 3-8 (cont.)
Filename Block Error Conditions

Error Code In File-name Block	Request Type	Cause	Remedy
14	.APPND .DELET .RENAM	An attempt was made to perform an illegal operation on an opened file.	Wait until file is closed.
15	.ALLOC .OPENO	An attempt was made to create a file with an illegal file name.	Change file name.
16	.RUN	All datasets were not released prior to issuing the request.	Release all datasets which were INITed.
17	.RUN	Load module format error.	File must be linked into a load module.
2Ø	.RUN	Specified CIL entry not found.	Add proper entry to CIL or use correct name.
21	.RUN	No transfer address or illegal transfer address.	Check for END statement in source program, or use correct /TR when linking.
22	.RUN	Stack base entry in the System Vector Table (SVT) is below the Stack Pointer. Stack cannot be moved as requested in the call.	
23	.RUN	Module is outside the boundaries of the allowable load area.	Relink to within boundaries. Ensure that resident portion of program is not being overlayed.

LINE BUFFER HEADER

4.3 THE LINE BUFFER HEADER - (used by READ and WRITE requests)

BUFHDR:

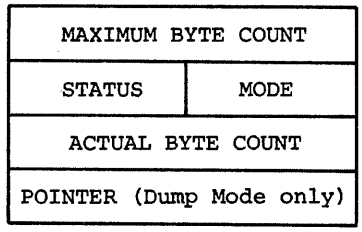


Figure 3-9
Line Buffer Header

Each element of the line buffer header table is as follows:

Address	Name	Function
BUFHDR	MAXIMUM BYTE COUNT	The count shows the size of the buffer, in bytes. It must be specified here by the user on all INPUT operations.
BUFHDR+2	MODE	The user specifies here the mode of the transfer. All modes are listed and explained in Figure 3-10.
BUFHDR+3	STATUS	The Monitor will place in this byte the status of the transfer when control is returned to the user. Figure 3-11 lists each bit and its meaning. Errors encountered executing an I/O transfer will be flagged in this byte. The user should always check its content after each transfer completes.
BUFHDR+4	ACTUAL BYTE COUNT	This count controls the number of bytes to be transferred on OUTPUT. It must be initialized by the user before any output transfer from the line buffer. After any transfer in or out, it will show how many bytes have been transmitted.
BUFHDR+6	POINTER (dump mode)	If bit 2 of MODE is 1, the user specifies here the starting address of the line buffer. If bit 2 of MODE is Ø, the line buffer header is only three words in length, and must immediately precede the line buffer itself.

NOTE

The user should not attempt to change the block or buffer contents until it is evident that the transfer has been completed. (e.g., after a .WAIT return). This is because the Monitor returns control to the program if a device transfer is needed to satisfy a request. During this time, the header words are used to store data relevant to the operation underway.

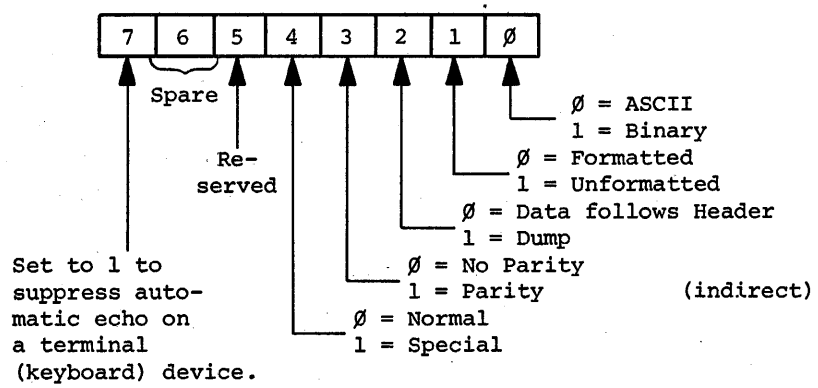


Figure 3-10
The Mode Byte

4.3.1 The Transfer Modes

The user can specify ASCII or binary data in nine different types of transfers:

ASCII Modes: Formatted ASCII Parity - Special
Formatted ASCII Parity - Normal

Formatted ASCII Nonparity - Special
Formatted ASCII Nonparity - Normal

Unformatted ASCII Parity - Special
Unformatted ASCII Nonparity - Normal

Binary Modes: Formatted Binary - Special
Formatted Binary - Normal

Unformatted Binary - Normal

1. Formatted ASCII Normal - Data in this mode is assumed by the Monitor to be in strings of 7-bit ASCII characters terminated by LINE FEED, FORM FEED, or VERTICAL TAB.

   READ: The line buffer is filled until either a terminator is seen or the number of bytes transferred becomes equal to the MAXIMUM BYTE COUNT. If the MAXIMUM BYTE COUNT is reached before the terminator is seen, the invalid line error bit in the Status Register of the buffer header is set, and each remaining character through to the terminator is read into the last byte of the line buffer, i.e., the surplus bytes are overlayed. After the transfer, the actual byte count is set to the number of bytes read (including the excess). RUBOUTs and NULLs are discarded. The terminator is transferred. LINE FEED is supplied after RETURN.

   WRITE: The line buffer is output until the number of bytes transferred equals the ACTUAL BYTE COUNT. If the last character is not a terminator, the invalid line error bit is set in the STATUS BYTE of the buffer header. Previous terminators are output as normal characters.

   For non-file-structured devices, TABs are automatically followed by RUBOUTs; FORM FEEDs are automatically followed by NULLs.

   The READ/WRITE processor passes data to the device driver specified, and each driver will convert the information to meet its specific needs. Appendix H summarizes the characteristics of the device drivers (see Part 5 for more information). Normally, output is deferred until the current buffer is full or until a .CLOSE or .RLSE occurs. However, for terminal devices, the buffer is written when a line terminator is seen. VERTICAL TAB plays a special role here, since it is a terminator but does not cause a carriage return or paper motion.

2. Formatted ASCII Special -

   READ: The same as formatted ASCII normal with this exception: if the MAXIMUM BYTE COUNT is reached before the terminator, the transfer is stopped. The remaining characters are not overlaid, but are retained for transfer at the next .READ. An invalid line error will be returned in the STATUS BYTE, and ACTUAL BYTE COUNT will equal MAXIMUM.

WRITE: The same as formatted ASCII normal with this exception: the line buffer is output until the first terminator; the ACTUAL BYTE COUNT will stop the transfer if it is reached before the terminator is seen. In this case, the invalid line error bit is set in the STATUS BYTE. Note that in this mode only one line of data can be output at once, but its byte count need not be exactly specified, provided it is not greater than the ACTUAL BYTE COUNT.

3. Formatted Binary Normal -

READ: This is an 8-bit transfer. Words 2 and 3, STATUS/MODE, and ACTUAL BYTE COUNT always accompany the data during formatted binary transfers. The counts are adjusted by the Monitor to include the extra words. On input, the line buffer is filled until the number of characters transferred equals the ACTUAL BYTE COUNT read, or the MAXIMUM BYTE COUNT. If the MAXIMUM is reached before the ACTUAL, an invalid line error occurs and the remaining bytes are overlaid into the last byte until the checksum is verified. After the transfer, the ACTUAL BYTE COUNT contains the actual number of data bytes read (including the excess).

WRITE: This is an 8-bit transfer. Words 2 and 3 of the line buffer header are output, and data is transferred until the number of characters transferred is equal to the ACTUAL BYTE COUNT; then a checksum is calculated. The checksum is output at the end. The byte count is adjusted to reflect the presence of words 2 and 3 from the line buffer header.

4. Formatted Binary Special -

READ: The line buffer is filled until the number of characters transferred equals the ACTUAL BYTE COUNT read. If the MAXIMUM COUNT is reached before the ACTUAL, the remainder of the line is retained by the Monitor. The MAXIMUM BYTE COUNT is transferred to the line buffer and the ACTUAL BYTE COUNT is set to the full input count, rather than to the number of bytes actually transferred. The invalid line error will be set in the STATUS BYTE. The user can compare the MAXIMUM COUNT with the ACTUAL, determine how much data remains, and recover it by an unformatted binary read (allowing 1 extra byte for the checksum).

WRITE: Identical to formatted binary normal.

5. Unformatted ASCII Normal or Special - This mode is available to the user who wants to do his own formatting. Seven bits are transferred; the eighth is always set to zero. Nulls are discarded.

READ: Transfer stops when the number of bytes transferred reaches the MAXIMUM BYTE COUNT. Nulls are discarded but all other characters are treated as valid.

WRITE: All characters are transferred. The transfer stops when the ACTUAL BYTE COUNT is reached.

6. Unformatted Binary Normal or Special - This mode is identical to unformatted ASCII except that eight bits are transferred on both input and output and nulls are not discarded. No checksum is calculated.

7. Formatted ASCII Parity - Identical to formatted ASCII (Special or Normal) except that even parity is generated in the eighth bit on OUTPUT; during INPUT it will be checked. Valid characters will be passed to the user as 7 bits; invalid characters will be marked by bit 8 = 1, and will cause the setting of the parity error bit in the STATUS BYTE.

8. Unformatted ASCII Parity - Identical to unformatted ASCII (Special or Normal) except that eight bits are transferred instead of seven. No parity generating or checking is performed.

9. Indirect Modes - All moves can be specified as indirect, which means that the word after ACTUAL BYTE COUNT is considered to be a pointer to the beginning of the data rather than the beginning of the data proper. This is referred to as DUMP mode.

4.3.2 The Status Byte

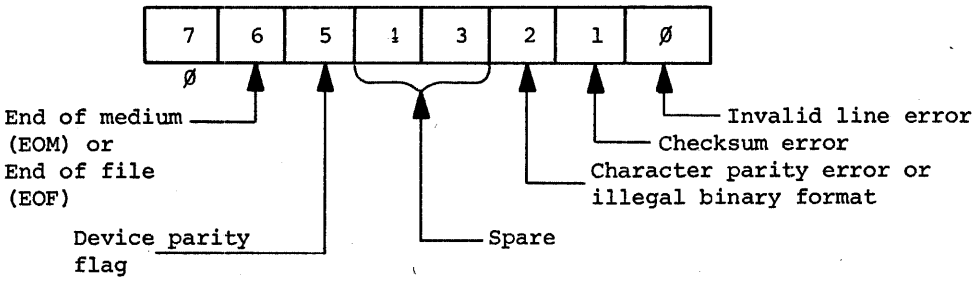


Figure 3-11
Status Byte Format

The function of each status format bit is explained below.

Bit	Mode	Request	Condition Bit = 1 indicates
	ALL	.READ/WRITE	Appropriate BYTE COUNT = Ø at call.
Ø (INVALID LINE)	FORMATTED ASCII NORMAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT was reached before a line terminator was seen. (Last byte has been overlaid until the terminator has been reached.)
		.WRITE	The last byte was not a terminator.
	FORMATTED ASCII SPECIAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT was reached before a line terminator was seen (excess data has not yet been read).
		.WRITE	The ACTUAL BYTE COUNT was reached before any terminator was seen.

Bit	Mode	Request	Condition Bit = 1 indicates
	FORMATTED BINARY NORMAL	.READ	The MAXIMUM BYTE COUNT was reached before the records internal byte count was exhausted. (The last byte has been overlaid in order to verify the checksum.)
	FORMATTED BINARY SPECIAL	.READ	The MAXIMUM BYTE COUNT was reached before the records internal byte count was exhausted. (The excess data still remains to be read and checksum has not been verified.)
	ALL UNFORMATTED MODES	.READ	BYTE COUNT = the actual number of bytes transferred. The reason BYTE COUNT < MAXIMUM BYTE COUNT is that an EOF or EOM has been encountered before the buffer was full. Bit 6 will also be set.
1 (CHECKSUM ERROR)	FORMATTED BINARY	.READ	There was a discrepancy between the checksum accumulated during the .READ, and that stored with the incoming data.
2 (PARITY FORMAT)	FORMATTED ASCII PARITY NORMAL OR SPECIAL	.READ	A character was read which had odd parity. The eighth bit of the illegal character delivered is set to a 1. The transfer continues. If this bit is set the user need only check each character returned during processing of the buffer for bit 8 set to locate the character returned with wrong parity.
2 (ILLEGAL BINARY FORMAT)	FORMATTED BINARY	.READ	This bit is set if a line processed in a binary mode does not have a ØØ1 in the first word. The first word is ignored, i.e., no data is returned to the buffer. Subsequent reads access successive lines and return error bits or data as appropriate.
6 (EOM/EOF)	ALL MODES	.READ or .WRITE	An input device cannot supply any more data or an output device cannot accommodate more, i.e., the disk has no more storage space, or the paper tape reader has run out of paper tape. No data is returned on .READs unless bit Ø is also set (see bit Ø). On .WRITEs an unspecified portion of the buffer may have been written (enough data to fill a partially filled monitor buffer may have been transferred to the buffer and written before the EOM or EOF was detected). Subsequent requests return to user with this bit set.

Bit	Mode	Request	Condition Bit = 1 indicates
5 (DEVICE PARITY)	ALL MODES	.READ or .WRITE	A hardware error has been detected on a bulk storage device. This could be either a parity error or a timing error. The driver will already have tried to READ or WRITE 8 or 9 times before setting this bit. (This flag is a warning that the data in this line or some subsequent line still using data from the same device block may be invalid. It will be returned for each transfer call using the same block.)

RECORD Block

4.4 THE RECORD BLOCK

FUNCTION/STATUS
BUFFER ADDRESS
RECORD LENGTH
HI ORDER, RECORD #
LO ORDER, RECORD #

Figure 3-12
The Record Block

ADDRESS	NAME	FUNCTION
RECBLK	FUNCTION/STATUS WORD	BIT

Ø - Not used
1 - Record Output - Set by user
2 - Record Input - Set by user
3-8 - Not used

(Following bits set by Monitor)

9 - Illegal Function
1Ø - File is linked or device is not File-structured.
11 - Record requested lies outside the file.
12 - File not OPEN
13 - Protect code violation, Incorrect Open
14 - Not used
15 - Device parity error

The user may set only bits 1 or 2; error bits are set by the Monitor, and should be tested for by the user upon return from the request. The error bits are cleared by the Monitor when a .RECRD request is issued and are set as appropriate upon return from the Monitor.

ADDRESS	NAME	FUNCTION
RECBLK+2	BUFFER ADDRESS	The address of the user's buffer. The buffer must be large enough to contain a record of the length indicated in the next word, as the Monitor assumes that sufficient space is available and will overlay data stored below a buffer of insufficient length.
RECBLK+4	RECORD LENGTH	The number of bytes of a Record. This value, which must remain the same for all records in the file, is supplied by the user.
RECBLK+6 RECBLK+1Ø	High Order - Record Number Low Order - Record Number	This entry identifies the record to be read or written. It is treated as a 32-bit number in anticipation of files that contain more than 65,536 records. First Record of File is number Ø.

4.5 THE BLOCK BLOCK - (used by BLOCK request only)

BLOCK Block

BLKBLK:

FUNCTION/STATUS
BLOCK NUMBER
MEMORY BUFFER ADDRESS
LENGTH

Figure 3-13
The BLOCK Block

Address	Name	Function
BLKBLK	FUNCTION/STATUS	User specifies here the function to be performed, and the Monitor returns to the user with the appropriate status bits set.

Type	Bit	Bit = 1 means:
function	Ø	function is GET
	1	function is OUTPUT
	2	function is INPUT
	3-8	reserved
error	9	illegal function
	1Ø	file is linked, or device is not file-structured
	11	block number does not exist in file, i.e., it is greater than the file length

Address	Name	Function
		Type: Status — Bit: 12 — Bit = 1 means: file not open
		Type: Status — Bit: 13 — Bit = 1 means: protect code violation
		Type: Status — Bit: 14 — Bit = 1 means: end of data error
		Type: Status — Bit: 15 — Bit = 1 means: device parity error
BLKBLK+2	BLOCK NUMBER	Requested block number to be transferred relative to the beginning of the file. First block of file is Ø.
BLKBLK+4	MEMORY BUFFER ADDRESS	The address of the buffer (supplied by the Monitor on INPUT or GET functions).
BLKBLK+6	LENGTH	The length of the buffer in words. BLKBLK+6 is set by the Monitor on INPUT or GET functions.

TRAN BLOCK

4.6 THE TRAN BLOCK (used by TRAN request only)

TRNBLK:

DEVICE BLOCK NUMBER
MEMORY START ADDRESS
POSITIVE WORD COUNT
FUNCTION/STATUS
NUMBER OF WORDS NOT TRANSFERRED

Figure 3-14
The TRAN Block

The user must set up a TRAN block before each .TRAN request in his program. See Section 3-3.6.41 for more information.

Address	Name	Function
TRNBLK	DEVICE BLOCK NUMBER	User specifies here the absolute block number of the device, at which the transfer is to begin. Block Ø is the first block on bulk storage devices. If it is not a bulk storage device, specify block Ø.
TRNBLK+2	BUFFER ADDRESS	User specifies here the core memory address at which the data transfer is to begin.
TRNBLK+4	WORD COUNT	User specifies here the total number of 16-bit words to be transferred. Word count may be more or less than block size.

Address	Name	Function
TRNBLK+6	FUNCTION/STATUS	Bit — Bit Meaning
		Ø — Binary = 1 - set by user ASCII = Ø - set by user
		1 — Write = 1 - set by user
		2 — Read = 1 - set by user
		3-1Ø — Reserved
		11 — DECtape direction - set by user Ø = forward 1 = reverse
		12 — Reserved
		13 — Invalid call (improper function/no word count)[1]
		14 — End of medium[1]
		15 — Recoverable device error (such as parity, timing, or record length)[1]
TRNBLK+1Ø	NUMBER OF WORDS NOT TRANSFERRED	User leaves this entry blank. If an EOM occurs during the transfer, the Monitor will place in this entry the number of words not transferred.

4.7 THE RUN BLOCK

RUN Block

The RUN Block is used exclusively with the .RUN request. It is a variable length control block containing a function word and several parameter words. The function word is always present; any of the parameter words may be omitted, depending upon the settings in the function word. Omitting a parameter word does not mean setting it to zero, but rather leaving it out. Hence, no parameter word occupies a set position in the RUN Block and the block itself is of variable length. For reference, all words but the function word are referred to by a letter, not by a number.

Table 3-9
Key to RUN Block Parameter Word

Word*	Parameter	Present If:
1	FUNCTION WORD	always
A	FILE BLOCK POINTER	Bit 15=Ø
B	LINK BLOCK POINTER	Bit 15=Ø
C	NAME	Bit 15=1 or Bit 13=1
D	NAME	Bit 15=1 or Bit 13=1
E	LOAD ADDRESS	Bit 3=1
F	TRANSFER ADDRESS OFFSET	Bit 4=1
G	RETURN ADDRESS	Bit 5=1

* Words A through G are so designated because any of them might be omitted under certain conditions.

[1]This bit is cleared by the Monitor upon .TRAN request issue and is set as appropriate upon return.

Address	Name	Function
RUNBLK	FUNCTION	User specifies here the function to be performed (see below).
RUNBLK+A	FILE BLOCK	Address of the File Block describing the file which contains the load module or core image to be loaded.
RUNBLK+B	LINK BLOCK	Address of the Link Block which describes the device from which the entity is to be loaded. Sufficient room must be provided in the Link Block to contain the EMT numbers of all Monitor modules which are to be loaded (these are contained in the load module, if there are any).
RUNBLK+C and RUNBLK+D	NAME	Two Radix-5Ø words containing either the name of the specific core image to be loaded from a CIL (bit 13=1) or the name of the file to be loaded if no File Block was given (bit 15=1)
RUNBLK+E	LOAD ADDRESS	Specifies an address at which the entity is to be loaded, without regard to the load address in the load module or CIL. The entity should be position independent.
RUNBLK+F	TRANSFER ADDRESS OFFSET	Specifies a value to be added to the transfer address obtained from the load module or CIL. Provides for alternate entry points to the module.
RUNBLK+G	RETURN ADDRESS	Specifies an address to which control must be passed when loading is completed. This address may or may not be in the loaded entity.

Figure 3-15
The RUN Block Description

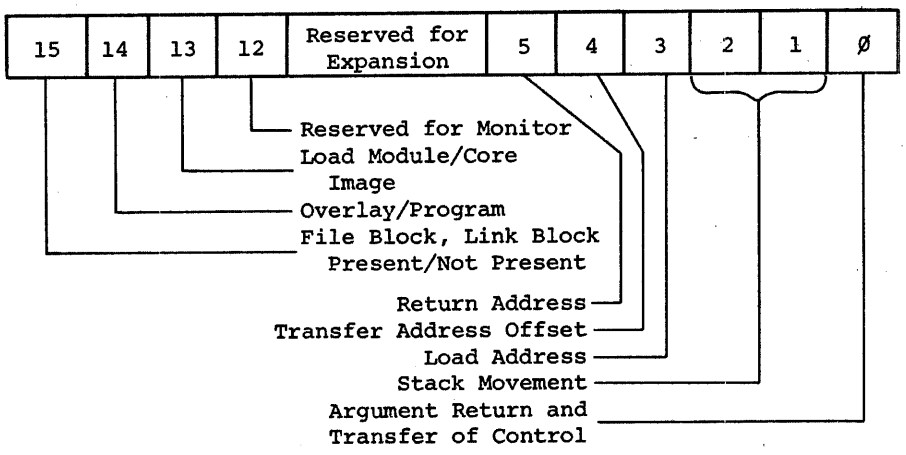


Figure 3-16
The RUN BLOCK Function Word

Bit Ø — Argument Return and Transfer of Control

= Ø Indicates control is to be returned to the instruction following the .RUN request after completing the requested actions, unless bit 5=1. Regardless of the setting of bit 5, the load module's transfer address, size in bytes, and low address will be on top of the stack when bit Ø=Ø (see Section 3-3.6.33).

= 1 Indicates control is to be switched to the transfer address of the loaded module after completion of the load, unless bit 5=1. Regardless of the setting of bit 5, no information is returned on the stack when bit Ø=1, but information may be passed by the call to the loaded module either on the stack or in the general registers.

Bit 1 — Stack Movement

= Ø Indicates that the stack is not to be moved from its present position under any condition.

= 1 Indicates that stack relocation may be necessary and that bit 2 of this word must be tested to determine under what conditions relocation will be necessary. The stack is never moved on short form calls (bit 15=1).

Bit 2 — Movement Condition

= Ø Indicates that the stack is to be unconditionally moved to the area directly below the module to be loaded. In this position the stack base entry in the System Vector Table (SVT) will be the same as the low address of the loaded module. The stack is never moved on short form calls (bit 15=1).

= 1 Indicates that the stack is to be conditionally moved, based on the relative positions of the stack base and low address of the module to be loaded. If the stack base entry in the SVT is higher than the low address of the module to be loaded, then the stack should be relocated as described above. If the stack base entry in the SVT is lower in core or equal to the low address of the module to be loaded, then the stack will not be relocated.

Bit 3 — Load Address

= Ø Indicates that no optional load address is specified in the RUN Block. The load address information in the load module will be used.

= 1 Indicates that the address specified in the RUN Block is to be used as the load address for the requested module. This entry overrides the load module information.

Bit 4 — Transfer Address Offset

= Ø Indicates that no offset from the module's transfer address is included in the RUN Block.

= 1 Indicates that the user desires an offset, specified in the RUN Block, to be added to the loaded module's transfer address. This offset is added to the transfer address regardless of the setting of bit Ø of the action word.

Bit 5 Return Address

= Ø Indicates that no alternate return address is included in the RUN Block. Return of control will thus be determined by the setting of bit Ø.

= 1 Indicates that an alternate return address has been specified in the RUN Block and that this address will receive control instead of the address following the .RUN request or the transfer address of the load module. The setting of bit Ø will still determine whether information will be returned on the stack.

Bit 12 Reserved for Monitor

= Ø This bit should always be zero.

Bit 13 Load Module/Core Image

= Ø Indicates that the entity being loaded is a load module. If the file identified by the File Block is a CIL, the first member of the CIL will be loaded.

= 1 Indicates that the entity to be loaded is a member of Core Image Library. The File Block identifies the CIL, while words 4 and 5 of the RUN Block contain the name of the CIL member.

Bit 14 Overlay/Program

= Ø Indicates that an overlay is being loaded. Since this is a continuation of the current program, datasets may be left open across this call. The overlay may not extend above the low address of the resident module, nor may it extend below the top of the Monitor area. System control tables are not refreshed as a consequence of this call. No additional Monitor modules may be made resident.

= 1 Indicates that a new program is being loaded. This is as if a new program were being RUN from the keyboard. Although all datasets must be released by the program which called RUN, RUN itself will do several things to refresh the environment. This includes releasing Monitor modules made resident by the previous program, undoing dataset assignments made specifically for the previous program, loading any Monitor modules which should be resident for this program, and changing any program-related values in the SVT.

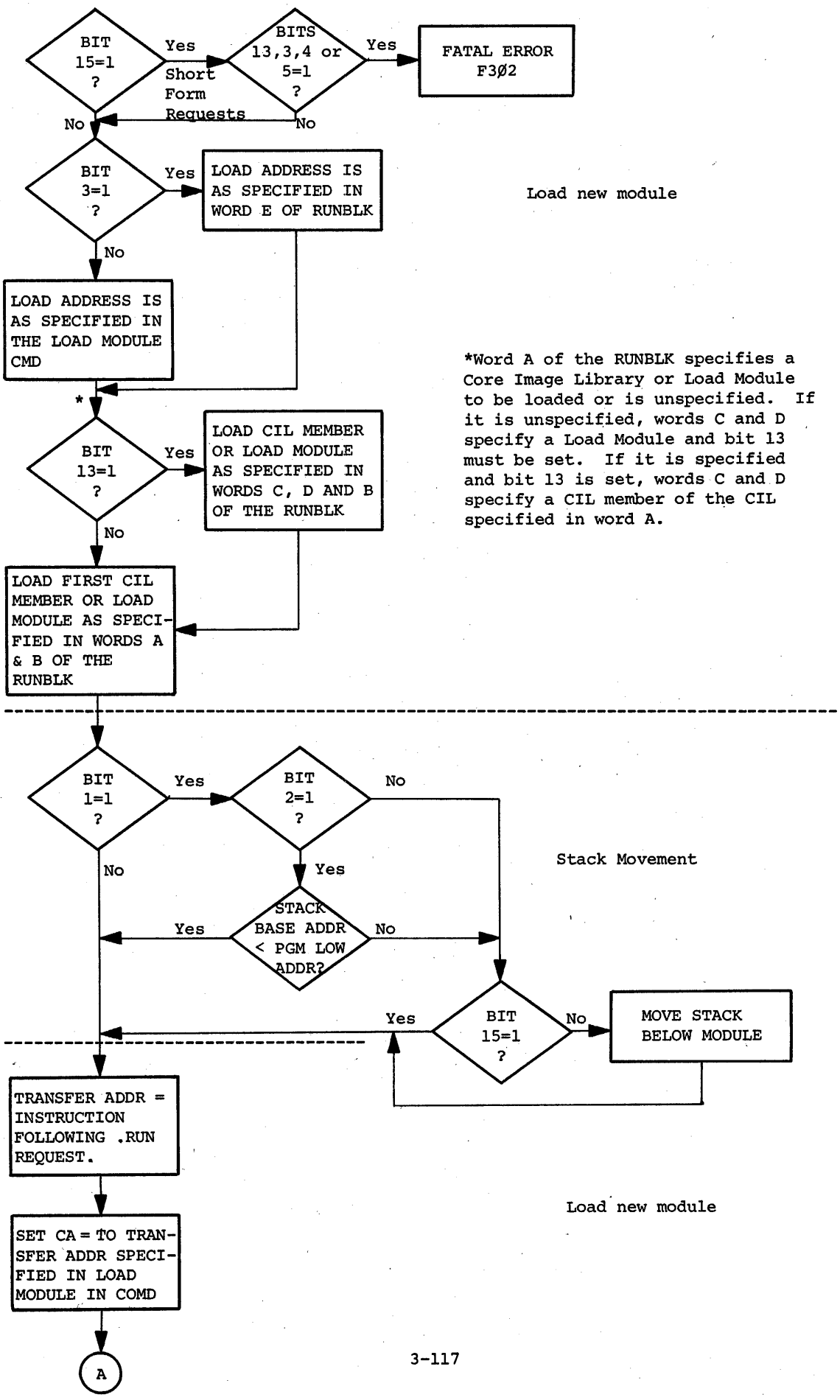
Bit 15 File Block, Link Block

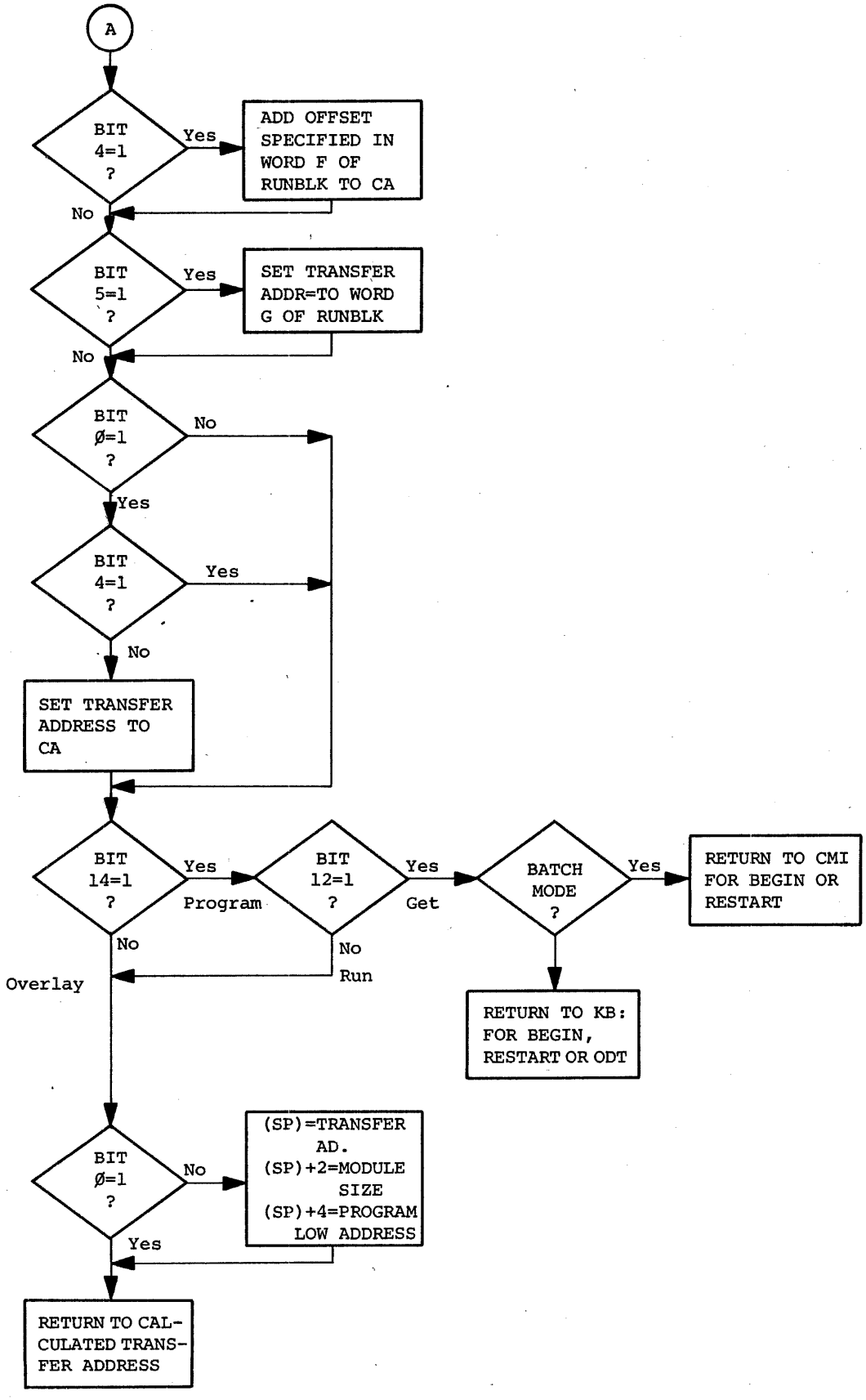
= Ø Indicates that a Link Block and a File Block pointer are in the RUN Block.

= 1 Indicates that the caller has provided a short form of the RUN Block; the short form contains only a function word and a six-character filename. The Link Block and File Block are created by the .RUN request itself. The entity to be loaded must be either in the current user's area or in the [1,1] UIC area and must have an extension of LDA or null. All other function bits are ignored. The load module or core image (first member of CIL) is loaded at its normal load address, as if it were an overlay, and receives control at its normal transfer address. The stack is not moved.

The following flowchart illustrates the processing of the function word bits.

RUN BLOCK FUNCTION WORD PROCESSING




A
BIT 4=1 ?
Yes
ADD OFFSET SPECIFIED IN WORD F OF RUNBLK TO CA
No
BIT 5=1 ?
Yes
SET TRANSFER ADDR=TO WORD G OF RUNBLK
No
BIT Ø=1 ?
No
Yes
BIT 4=1 ?
Yes
No
SET TRANSFER ADDRESS TO CA
BIT 14=1 ?
Yes
Program
BIT 12=1 ?
Yes
Get
BATCH MODE ?
Yes
RETURN TO CMI FOR BEGIN OR RESTART
No
Overlay
No
Run
RETURN TO KB: FOR BEGIN, RESTART OR ODT
BIT Ø=1 ?
No
(SP)=TRANSFER AD.
(SP)+2=MODULE SIZE
(SP)+4=PROGRAM LOW ADDRESS
Yes
RETURN TO CALCULATED TRANSFER ADDRESS

PART 3

CHAPTER 5

SUBSIDIARY ROUTINES AND OVERLAYS

With the exception of .READ/.WRITE and .WAIT, all Monitor code for performing programmed requests is potentially non-resident. Since non-resident modules are limited to a size of 256 words (the size of the swap buffer) and since many common functions are required, many of the programmed request modules must make use of subsidiary routines. Table 3-10 can be used in two ways:

1. The table shows how many modules (in addition to the primary module) may be loaded to satisfy a particular type of request.

2. When making certain functions resident, the user should make the primary module resident, and also each of the subsidiary modules which may be called. For example, if the user wants all .OPENI processing routines (except for magtape) resident, he would put the following assembler directive in his program:

```
.GLOBL OPN.,FOP.,LUK.,CKX.
```

The following summary explains the codes used in the table.

(blank) = subsidiary routine is never called

X = subsidiary routine is called only when a file-structured device is referenced

L = subsidiary routine is called only when a linked file is referenced

C = subsidiary routine is called only when a contiguous file is referenced

D = subsidiary routine is called only when DECtape is referenced

M = subsidiary routine is called only if magtape is referenced

T = subsidiary routine is called only if cassette is referenced

Table 3-10
EMT Service Routines

Global Name of Primary Module	Request	Name of Subsidiary Routine: Open Existing File (FOP.)	Creating New Linked File (FCR.)	Close File (FCL.)	Directory Search (LUK.)	Allocate Block, Linked File (LBA.)	Get Bit Map Segment (GMA.)	Allocate Contiguous Blocks (CBA.)	Check Access (CKX.)	Delete Linked File (DLN.)	Delete Contiguous File (DCN.)	Append DECtape (AP2.)	Get Next Bit Map Segment (GNM.)[2]	Magtape Open (MTO.)	Loader (LDR.)[4]	Loader (LD2.)[4]	Cassette Tape Open (CTØ.)
RWN.	.READ/WRITE[1]												X				
OPN.	.OPENU	X			X				X					M			T
OPN.	.OPENO[3]		X		X	X	X		X					M			T
OPN.	.OPENE	X			X	X	X		X					M			T
OPN.	.OPENI[4]	X			X				X					M			T
OPN.	.OPENC	X			X				X								
CLS.	.CLOSE[4]			X													
ALO.	.ALLOC				X			X	X								
DEL.	.DELET				X				X	L	C						
REN.	.RENAM				X				X								
APP.	.APPND				X				X			D					
DIR.	.LOOK				X				X								
RUN.	.RUN[4]	X		X	X				X					M	X	X	T
INR.	.INIT[5]																
RLS.	.RLSE[4]																

[1]Always resident.

[2]Should never be made resident.

[3]The .OPENO module requires a second section if a dataset other than CMO is being opened on the device assigned to CMO.

[4]The .RUN EMT calls the following routines:

```
.INIT
.OPENI      (once for each combination of filename and UIC)
.LDR        (three sections if LDA file; two if CIL file)
.LD2
.CLOSE      (once for each .OPENI)
.RLSE
```

[5]The .INIT module has two sections, but the second has no name. It is resident automatically if .INIT is resident.

PART 3

CHAPTER 6

COMMAND STRING INTERPRETER

6.1 SYSTEM PROGRAM/USER PROGRAM COMMAND STRINGS

There is a single, general format for all system program command strings. All system programs use it, and any user program may also do so. These command strings are all processed by a Monitor routine, the Command String Interpreter (CSI) which is in Section 3-3.6.8. Any program expecting such a command first types # on the console to indicate the fact to the operator. The general format is

ds-spec[,ds-spec]...][<[ds-spec][,ds-spec]...]

where "ds-spec" represents a dataset specification.

6.2 CSI COMMAND FORMAT

Whenever a system program requests input through the CSI, a # will be printed on the terminal and the program will wait for the operator's reply. A CSI command may consist of one or more output dataset specifications, followed by <, followed by one or more input dataset specifications. Spaces, horizontal TABs, and nulls may appear anywhere in the string and are ignored. A command is terminated by typing the RETURN key, which causes both carriage return and line feed characters to be passed to the program. The line-feed character terminates the input. < need not occur. If it does, at least one input file specification must appear. Only one < per command is allowed. Commands cannot be continued from line to line.

A dataset specification must be delimited by a comma. If no items appear before the comma, it is interpreted as "this particular positional field will not be used". For example, suppose a program requires three (output) data specifications. Then the syntax:

Dataset Specification,,Dataset Specification

indicates that the second (output) dataset specified will not be generated.

Each dataset specification is a field which describes a dataset. It generally contains information as to where to find the dataset, the file name and extension if the dataset is a file, the user identification code associated with the file, and one or more switches which request various actions to be performed. A dataset specification containing all of the above elements would appear as:

dev:filnam.ext[uic]/sw_1:v_1:...:v_n/sw_2:v_1:...:v_n,

where: dev = The device specification consisting of two or three letters (and often an octal digit) terminated by a colon. The letters identify the device and the digit identifies the unit. Units must be given in octal. The colon delimits this field with one exception; only physical names as listed in Appendix A may be specified. For example, DTA1: is the correct specification for DECtape, controller A, unit 1. The exception is SY: which is a generic name for the system residence device (e.g., on an RK system SY: is equivalent to DK:). If no digit appears, unit Ø is assumed. If the device specification itself does not appear, the device is assumed to be the last device specified on the current side of the <, if there is one; otherwise, the system disk (SY:) unit Ø is assumed.

filnam = The file name specification consists of one or more letters or digits, or exactly one asterisk. The first six letters or digits specify the name. The first character must be a letter. All letters and digits in excess of six are ignored.

The file name need not appear if the device is not file-structured or if the program can supply a name.

.ext = The extension specification consists of a period, followed by one or more letters or digits, or followed by exactly one asterisk. The first three letters or digits specify the extension. All letters or digits in excess of three are ignored.

The extension need not appear.

The asterisk is used to specify "all". For example:

*.EXT specifies all files with extension .EXT,
FIL.* specifies all files with name FIL, and
. specifies all files and all extensions.

[uic] = The User Identification Code (UIC) specification consists of a left square bracket, followed by one or more octal digits or exactly one asterisk, followed by a comma, followed by one or more octal digits or exactly one asterisk, followed by a right square bracket. The field to the left of the comma specifies the user's group and the field to the right of the comma specifies the user within the group. Both fields must be given in octal, and the largest valid octal number is 376 in both cases (Ø is invalid). For example, [12,136] is the correct specification for user number 136 of user group 12.

NOTE

The left and right square brackets are not visible on some keyboard keys; however, they may be typed using SHIFT/K and SHIFT/M, respectively.

As in filnam and .ext, the asterisk specifies "all". For example:

[*,136] specifies all users whose number is 136
[12,*] specifies all members of user group 12, and
[*,*] specifies all users.

The user identification code need not appear, in which case the default is the identification entered with the LOGIN command.

/sw:v_1:...:v_n = A switch specification consists of a slash (/), followed by one or more letters or digits, and optionally followed by one or more value specifications. A value specification is initially delimited by a colon. The value itself can be null, or consist of one or more letters, digits, periods, or dollar signs. Other characters are illegal. The digits 8 and 9 are legal.

For examples: /DATE:12.2Ø.69 might be a switch to enter December 2Ø, 1969 in a date field.

/DATE:12::69 might enter December, 1969 in a date field.

Switches need not appear. If a switch does appear, it need not contain more than one letter or digit after the slash. For example:

/S and /SWITCH2 are both legal.

The first two characters after the slash uniquely identify the switch. For example:

/S is treated as if it were /S null.
/SWITCH1 and /SWITCH2 are both treated as /SW.

Table 3-11 summarizes the legal command syntax.

Table 3-11
.CSI Command String Syntax Rules

Item Which Last Appeared	Item Immediately Following								
	,	DEV:	FILNAM	.EXT	UIC	/SWITCH	<	Terminator	*
blank[1]	*	*	*	E	*	*	*	*	*
,	*	*	*	E	*	*	*	*	*
DEV:	*	E	*	E	*	*	*	*	*
FILNAM	*	E	E	*	*	*	*	*	E[2]
.EXT	*	E	E	E	*	*	*	*	E
UIC	*	E	E	E	E	*	*	*	E
/SWITCH	*	E	E	E	E	*	*	*	E
<	*	*	*	E	*	*	E	E	*

Legend: E indicates error. * indicates legal.

[1]The next item encountered is the first item in the command string.

[2] .* is legal following FILNAM.

For example, a device specification immediately followed by an extension specification is an error, whereas a file name specification immediately followed by a comma is legal. Note that a /SWITCH specification is always legal even alone. In such a case, the system device SY: and a null filename are assumed.

6.3 CSI COMMAND EXAMPLE

An example of a complete command is:

```
F1.E1,,DTA1:F2.E2/S:1<F3.E3[11,123],DTB:F4.E4/AB,F5.E5
```

which is interpreted by CSI2 as explained below.

a. The first positional output dataset is to be a file named F1 and will have extension E1. Its data device is the system device unit Ø, and catalogued under the ID of the user who entered the command. No switches are associated with this dataset.

b. The second positional output dataset will not be generated.

c. The third positional output dataset is to be in a file named F2 and will have extension E2. It is to be put on the DECtape which is mounted on unit 1 of controller A. This file is to be catalogued under the ID of the user who entered the command. The action indicated by switch S with value 1 is to be performed on this dataset.

d. The first positional input dataset is a file named F3, and its extension is E3. It can be found on the system device unit Ø, catalogued under UIC [11,123]. No switches are associated with this dataset.

e. The second positional input dataset is a file named F4, and its extension is E4. It can be found on the DECtape currently mounted on controller B, unit Ø. Associate the ID of the user who entered the command with this dataset. Perform the action indicated by switch AB on this dataset. No values are associated with the switch.

f. The third positional input dataset is a file named F5 and its extension is E5. It can be found on the DECtape currently mounted on controller B, unit Ø. Associate the ID of the user who entered the command with this dataset. No switches are associated with this dataset.

PART 3

CHAPTER 7

SPECIAL I/O FUNCTIONS

7.1 SPECIAL FUNCTION BLOCK AND CODE

Certain I/O functions are sufficiently device-dependent that they are not included within the scope of the general I/O facilities. The .SPEC request (see Section 3-3.6.3) is provided as a means of accommodating such functions. A special function request requires one argument, which must be either a code in the range Ø-255 or a pointer to a special function block. When a special function block is used, it must contain a code. See Figure 3-17.

7.1.1 The Special Functions Block (used for SPEC request only)

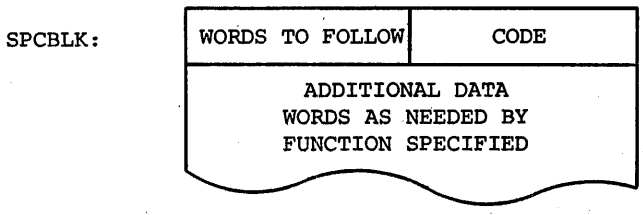


Figure 3-17
The Special Functions Block

Where a special function requires supporting data, the user must set up a Special Functions Block in his program.

Address	Name	Function
SPCBLK	CODE	The user identifies the function here by inserting the appropriate code in the range Ø-$255_{1Ø}$.
SPCBLK+1	WORDS TO FOLLOW	The size of each Special Functions Block is dependent upon the Function. The user shows here how many more words belong to the particular block.
SPCBLK+2 . . .	---	The user places in these words data to be passed to the function processor or the function processor will return here such items as status information, etc. The format in each case is determined by the function.

7.1.2 The Special Functions Code

Code	Function
1	Offline (rewind and unload)
2	Write End-of-File
3	Rewind
4	Skip Record(s)
5	Backspace Record(s)
6	Set Density and Parity
7	Obtain Status
8	Set Buffer Size
9	Rewind Enable/Disable
1Ø	Space Forward Files
11	Space Reverse Files
12	READ After Write Verification

In general, special function codes will have similar meanings from device to device. When a code has no meaning for a device, it is ignored.

7.2 MAGTAPE SPECIAL FUNCTIONS

7.2.1 Special Function Block

The magtape driver requires a special function block to perform special function requests. The following is the calling sequence for magtape special functions and the special function block format:

```
        MOV #SFBLK,-(SP)        ; Address of special function block
        MOV #LNKBLK,-(SP)       ; Address of link block
        EMT 12                  ; Special function EMT
        .
        .
        .
SFBLK:  .BYTE 3                 ; Special function code (e.g., rewind)
        .BYTE 3                 ; Words to follow (must be 3 or larger)
        .WORD Ø                 ; Tape unit status (returned by driver)
        .WORD Ø                 ; User specified count or control
                                ; information
        .WORD Ø                 ; Residue count (returned by driver)
```

7.2.2 Special Function Code

7.2.2.1 OFFLINE (Rewind and Unload - Function Code 1)

This request causes the magtape to be rewound to the beginning-of-tape (BOT) marker and SELECT REMOTE status to go off. If the last command to the driver for this device was a WRITE, three EOF's are written before rewinding. Thus, this function could cause data to be lost if it is issued before a CLOSE during READ/WRITE processing.

7.2.2.2 WRITE END-OF-FILE - Function Code 2

This request writes an end-of-file (EOF) record on magtape. It may cause data to be lost as described under OFFLINE.

7.2.2.3 REWIND - Function Code 3

The REWIND request performs the same function as OFFLINE except that the SELECT REMOTE status does not go off.

7.2.2.4 SKIP RECORD(S) - Function Code 4

Skips forward over the requested number of records (SFBLK+4) until either the SKIP count is exhausted or until an EOF record is encountered, in which case the EOF is spaced over and counted, but the operation terminates and a residue count (SFBLK+6) is returned (if any).

7.2.2.5 BACKSPACE RECORD(S) - Function Code 5

This request skips backwards over the requested number of records until either the SKIP count is exhausted or an EOF or the BOT marker is encountered. If an EOF is encountered it is spaced over and counted, but the operation terminates and a residue count is returned (if any). If the BOT marker is encountered, it is not skipped or counted. Instead, the operation is terminated and a residue count is returned.

7.2.2.6 SET DENSITY AND PARITY - Function Code 6

DENSITY (SFBLK+5)	PARITY (SFBLK+4)
Ø = 2ØØ BPI	Ø = ODD
1 = 556 BPI	1 = EVEN
2 = 8ØØ BPI	
3 = 8ØØ BPI Dump Mode	

The default density and parity are 8ØØ BPI Dump Mode, ODD. In this mode, one byte from core is represented as two bytes on 7-track magtape or one byte from core is represented as one byte on 9-track magtape. For 7-track tape, changing from this default causes one byte from core to be represented by one byte on tape with a loss of the two high order bits (6-7) of the byte.

7.2.2.7 TAPE UNIT STATUS - Function Code 7

This request returns the current status of the tape unit in SFBLK+2 in the following form:

Bits	Content
Ø - 2	Last command was: Ø = OFFLINE 1 = READ 2 = WRITE 3 = WRITE EOF 4 = REWIND 5 = SKIP RECORD 6 = BACKSPACE RECORD
3 - 6	Unused
7	1 = TAPE AFTER EOF (BEFORE EOF IF LAST COMMAND WAS BACKSPACE)
8	1 = TAPE AT BOT MARKER
9	1 = TAPE AFTER EOT MARKER
1Ø	1 = WRITE LOCK ON
11	PARITY: Ø = ODD 1 = EVEN (DEFAULT = ODD)
12	Ø = 9 TRACK 1 = 7 TRACK
13 - 14	DENSITY: Ø = 2ØØ BPI 1 = 556 BPI 2 = 8ØØ BPI 3 = 8ØØ BPI DUMP MODE
15	1 = LAST COMMAND CAUSED ERROR

Tape unit status is returned in SFBLK+2 for all special functions.

7.2.2.8 SET BUFFER SIZE - Function Code 8

Set Buffer Size performs either of the following functions:

1. Allows the actual byte count of the data to be transferred to be specified; the transfer is not restricted to an even byte count.

2. Allows specification of the actual byte count, allocates the buffer, and performs READ/WRITE I/O as specified.

When the first option is selected, the user must ensure that all transfers start on a word boundary. Option 1 applies only to TRAN I/O processing. The conditional parameter symbol RECORD must be undefined.

When Option 2 is selected, if a buffer has been allocated previously (DDB+6 ≠ Ø), S.RLB deallocates it and S.GTB allocates a buffer of the correct size. The DDB is updated. RECORD must be defined (RECORD = Ø). If sufficient memory is not available, an FØ77 error message is issued.

For either option, the actual byte count desired is specified as a positive value in the third word of the special function block.

The user must never specify a byte count less than 28 bytes (decimal). Because Magnetic Tape Open processor (EMT 63_8) requires at least 28 bytes, files with shorter counts cannot be OPENed or CLOSEd on magtape.

7.2.2.9 REWIND ENABLE/DISABLE - Function Code 9

Rewind Enable/Disable provides an explicit rewind attribute to the magtape driver. The driver retains the attribute until the specified unit is released using an EMT 7 or until enabled or disabled again. The driver refers to the specified unit's rewind attribute whenever it considers commanding a magtape drive (unit) to rewind. If the attribute is disabled, the driver does not issue the rewind command. If the attribute is enabled, the driver issues the rewind command.

The third word of the special function block contains the attribute in the following format.

Ø indicates that rewind is enabled.
nnn indicates that rewind is disabled.
nnn must be in the range 1 through 377_8.

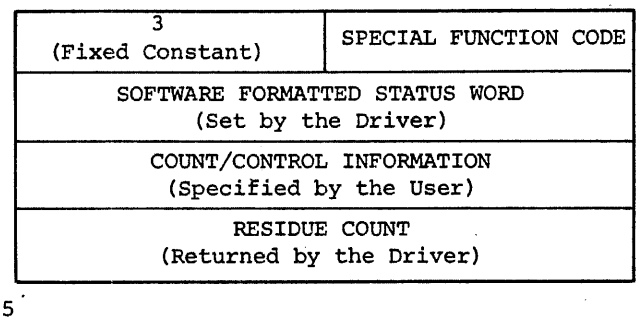
7.3 CASSETTE TAPE SPECIAL FUNCTIONS

Special functions for cassette tape provide the user with access to extended and/or optional capabilities implemented within the device driver or the actual peripheral system. Special function requests cannot be performed until the dataset to which they refer has been initialized using an EMT 6.

The use of special functions must be consistent with the file structure imposed on cassette tape. The user must be careful when using special functions because several of them reposition the tape or alter the record length.

7.3.1 Special Function Block

Cassette tape special functions must be requested using the special function block, which has the following format.



Unrecognized special function codes are ignored.

The following is an assembly language example of the use of the special function block.

```
        MOV #SFBLK,-(SP)        ; Push SFBLK's address
        MOV #LNKBLK,-(SP)       ; Push LNKBLK's address
        EMT 12                  ; Special function EMT
        .
        .
        .
        .
        .
SFBLK:  .BYTE   4               ; Special function code
        .BYTE   3               ; Number of words to follow (3)
        .WORD   Ø               ; Software formatted status
        .WORD  -5               ; User specified count
        .WORD   Ø               ; Residue count
```

The contents of these locations are not specified as a part of the calling sequence.

Encountering a special function block that does not contain the constant 3 in the second byte causes the driver to print an FØ33 diagnostic message on the console. The user's program is suspended.

7.3.2 Special Function Code

7.3.2.1 OFFLINE (Rewind and Unload) - Function Code 1

The OFFLINE special function rewinds the cassette; the cassette remains on-line. If a write function was the last command performed, OFFLINE causes an EOF and an appropriate sentinel label to be written before the rewind occurs. Otherwise, the only action is to rewind the cassette tape.

7.3.2.2 WEOF (Write End-of-File) - Function Code 2

The WEOF special function indicates that an end-of-file gap is to be placed on the cassette tape. Because adjacent file gaps cannot be detected, the WEOF special function is ignored if the last command performed was WEOF.

During READ/WRITE level transfers, data may be lost if a CLOSE is not performed before issuing a WEOF special function.

7.3.2.3 REWIND - Function Code 3

The REWIND special function indicates that the cassette tape mounted on the specified transport is to be rewound to BOT. If the last command performed was a write function, an EOF and a sentinel label record are written before rewinding.

During READ/WRITE level transfers, data may be lost if a CLOSE is not issued before the REWIND special function.

7.3.2.4 FBLOCK (Space Forward Blocks) - Function Code 4

The FBLOCK function indicates that the cassette tape is to be positioned forward the specified number of blocks (records). The user specifies the number of records to skip as a two's complement negative integer placed in the third word of the special function block.

Normally, expiration of a hardware timer terminates a space forward block command, thus indicating entry into a file gap. The exception to this rule occurs if a cassette is positioned at BOT (i.e., within the clear leader) and a space forward block command is executed. In this case, tape motion is initiated and is not terminated until the IRG (inter-record gap) after the first record. Performing FBLOCK on an uninitialized cassette or on a cassette containing only file gaps results in the tape moving until physical EOT. Blank cassettes must be initialized using the PIP /ZE switch.

The expiration of a hardware timer fails whenever FBLOCK encounters adjacent file gaps; i.e., two file gaps are detected and retrieved as three gaps.

The space forward blocks special function request is terminated if an EOF or EOT is encountered.

7.3.2.5 RBLOCK (Space Reverse Blocks) - Function Code 5

The RBLOCK special function indicates that the cassette tape is to be backspaced the specified number of blocks (records). The user specifies the number of

records to backspace as a two's complement negative integer placed in the third word of the special function block.

The RBLOCK request is terminated if it encounters BOT. It cannot detect EOF when backspacing.

RBLOCK must encounter data before it starts spacing into the IRG (inter-record gap) that is to terminate tape motion. Therefore, if a cassette is positioned in the IRG following a file gap and a request to backspace one record is issued, the tape is positioned into the IRG before the data record that precedes the file gap.

7.3.2.6 PARITY (Parity/Density) - Function Code 6

The PARITY special function is not implemented for cassette tapes; if this function code is encountered, the device driver exits immediately.

7.3.2.7 STATUS (Software Formatted Status) - Function Code 7

The STATUS special function retrieves the cassette tape peripheral system status register contents and places an evaluated and rearranged version of it in the second word of the special function block. The evaluation segregates the BOT from the EOT situation and examines the possibility that the last transfer encountered an EOF.

Figure 3-18 illustrates the information placed in the second word of the special function block by the STATUS request.

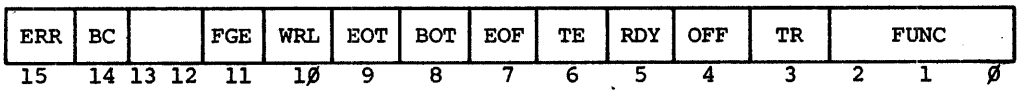


Figure 3-18
STATUS Information

ERROR (ERR) is a collective indicator of all possible errors and is set upon detection of any error within the TAll peripheral system. Error conditions are defined and evaluated by the TAll control module as a specific collection of abnormal relationships that can exist between the function code in bits 1 through 3 and that function's state as reflected in the status bits 4 through 14. ERROR is valid only when ready is set.

BLOCK CHECK (BC) indicates that the 16-bit cyclic redundancy check (CRC) character appended to the current block of data has failed inspection during a read function. BLOCK CHECK is cleared upon initiating a function or during initialization.

Initiating a function causes the setting of the GO bit and then the acceptance of the specified function code by the TU6Ø. Initialization is the execution of a RESET instruction, execution of the computer power-up sequence, or depression of the START switch on the computer console.

BLOCK CHECK is not altered upon selection of the other transport.

FILE GAP ENTERED (FGE) indicates that a file gap has been entered during the execution of any of the following functions:

- Read,
- Space reverse file,
- Space forward block,
- Space forward file.

FILE GAP ENTERED is cleared upon initiating a function or during system initialization. It is not altered by selection of the other transport.

WRITE LOCK (WRL) indicates that the cassette is write-protected and occurs only when a write or a write file gap function resides within the function code bits. WRITE LOCK reflects the momentary status of the selected transport (i.e., WRITE LOCK is updated upon selection of the other transport if either of the correct functions reside in the function code bits).

EOT indicates that clear leader or trailer has been detected after performing a function that moves the tape in a forward direction. The driver must not have been released.

BOT indicates that clear leader or trailer has been detected after performing a function that moves the tape in a reverse direction. The driver must not have been released.

EOF indicates either of the following conditions:

1. During the specified unit's last read TRAN processing, an EOF was encountered; i.e., an attempt was made to read a file gap,
2. During the specified unit's last CLOSE processing, rewind was specified and the tape was positioned at EOF.

In either case, the driver must not have been released.

Whenever EOF is set, the tape is positioned in the IRG immediately preceding the file gap.

TIMING ERROR (TE) indicates the loss of data during either a read or write function. TIMING ERROR results when the software response to the transfer request exceeds the transfer latency (1.8 milliseconds). It is cleared when a function is initiated or during system initialization.

TIMING ERROR is not altered by selection of the other transport.

READY (RDY) indicates the presence of power and that the selected TU6Ø is prepared to consider a function request. READY and TRANSFER REQUEST are mutually exclusive indicators. READY is cleared upon initiation of a function, setting of TRANSFER REQUEST, or completion of a function. READY is set during initialization.

OFF-LINE (OFF) indicates the loss of power at the TU6Ø or the lack of a cassette cartridge in the selected transport. OFF-LINE reflects the status of the selected transport and is updated upon selection of the other transport.

TRANSFER REQUEST (TR) indicates a demand for the system to perform a data byte transfer or initiate the transfer termination sequence. During a read function, TRANSFER REQUEST indicates that a data byte is to be retrieved from the TA11 DBR (7775Ø2). During a write function, TRANSFER REQUEST indicates that the data byte is to be placed in the TA11 DBR. Associated with TRANSFER REQUEST is a 1.8 millisecond latency window; if it expires data is lost.

TRANSFER REQUEST and READY are mutually exclusive. TRANSFER REQUEST prevents READY from being set, and once set, TRANSFER REQUEST demands servicing prior to READY. TRANSFER REQUEST is cleared upon addressing the TA11 DBR, setting INITIATE LAST BYTE SEQUENCE, or during system initialization.

FUNCTION (FUNC) specifies the operation to be performed. Throughout the execution of any function except rewind, the TU6Ø repeatedly refers to the function code to determine the following:

1. Operational state,
2. Motion direction,
3. Action definition,
4. Error definition.

Therefore, the function code must not be altered during execution. FUNCTION is cleared during initialization.

Table 3-12 defines the bit assignments for the function codes.

Table 3-12
Function Codes

Function	Code (Bit Assignment)	Mnemonic
Write File Gap	ØØØ	WFG
Write	ØØ1	WRITE
Read	Ø1Ø	READ
Space Reverse File	Ø11	SRF
Space Reverse Block	1ØØ	SRB
Space Forward File	1Ø1	SFF
Space Forward Block	11Ø	SFB
Rewind	111	REWIND

7.3.2.8 BLOCK (Record Length Specifications) - Function Code 8

The BLOCK special function enables the user to indicate a specific record length to the device driver for TRAN level transfer of records with an odd number of bytes: i.e., the data does not terminate on a word boundary. The TRAN block permits the user to specify only a word count.

In addition, if the device driver's RECORD conditional assembly option has been used, the user can indicate a specific record length to both the device driver and the file structure. This facility provides the user with READ/WRITE level transfer of records with any fixed length. This does not imply support of variable-length record transfer.

The user specifies the number of bytes per record as two's complement positive integer placed into the third word of the special function block.

BLOCK special function request servicing occurs independent of the cassette tape peripheral system.

7.3.2.9 GOVERN (Enable/Disable) - Function Code 9

The GOVERN special function provides an explicit rewind attribute for the purpose of regulating attempts to rewind the specified unit. Both OPEN and CLOSE processing execute rewind commands. The device driver retains each unit's rewind attribute until the attribute is explicitly altered or until the unit is released. The user places the rewind attribute in the third word of the special function block using the following conventions:

Ø = enable rewinds,

$1\text{-}377_8$ = disable rewinds.

The device driver refers to the specified unit's rewind attribute whenever considering the issuance of a rewind command. If disabled, the device driver does not issue the command. If enabled, the device driver issues the command. Unless explicitly altered, each unit's rewind attribute is enabled, and when any unit is released, the unit's rewind attribute is reset to enable rewinds.

7.3.2.10 FFILE (Space Forward Files) - Function Code 1Ø

The FFILE special function indicates that the cassette tape is to be positioned forward the specified number of files, i.e., file gaps. The user specifies the number of files to be skipped as a two's complement negative integer placed in the third word of the special function block.

The space forward files request is terminated if it encounters physical EOT.

If a cassette is positioned within the IRG following the last record of a file and a space forward file command is executed, the tape is positioned approximately two thirds the distance into the file gap.

Normally, expiration of a hardware timer terminates a space forward file command, thus indicating entry into a file gap. The exception to this occurs if a cassette is positioned at BOT (within the clear leader) and an FFILE command is executed. Then tape motion is initiated and is not terminated until the first file is skipped. Thus, performing FFILE on either an uninitialized cassette or a cassette containing only file gaps does not terminate until EOT. Blank cassettes must be initialized using the /ZE switch of PIP.

The expiration of a hardware timer technique does not work whenever adjacent file gaps are encountered; e.g., two file gaps are detected and retrieved as three file gaps.

7.3.2.11 RFILE (Space Reverse Files) - Function Code 11

The RFILE special function indicates the cassette tape is to be reverse positioned the specified number of files. The user indicates the number of files as a two's complement negative integer placed in the third word of the special function block. The space reverse files is terminated if BOT is encountered; i.e., clear leader is detected.

Under all circumstances, data must be encountered before a RFILE request will begin seeking the file gap that is to terminate motion. Specifically, if a cassette is positioned within an IRG following a file gap and RFILE is issued, the tape is backspaced into the file gap preceding the last file.

7.3.2.12 VCHECK (Read After Write Verification) - Function Code 12

The VCHECK special function request permits the user to enable the read after write verification so that he can ensure the retrievability of data transferred to the cassette. This optional feature requires use of the conditional assembly VERIFY symbol. The verification attribute is specified in the third word of the special function block as follows:

Ø = disable verification,
$1\text{-}377_8$ = enable verification.

The device driver refers to the appropriate unit's verification attribute when initiating an output transfer. Unless explicitly altered, each unit's verification attribute is disabled, and upon releasing a unit, the unit's verification attribute is reset to disable verification.

7.4 CARD READER SPECIAL FUNCTION (CODE 1)

Normally, the default conditions for the card reader are established automatically for the caller as part of the OPENing process. Under unusual circumstances, performing an OPEN may not be desirable or possible.

For example, the Operating System's Batch Stream Manager only OPENs the batch stream once for an unknown number of independent jobs, each of which expects the card reader's default condition to be established. If the batch stream has been assigned to the card reader, a problem arises.

Thus, the card reader special function has been implemented to establish the default conditions as follows:

1. Internal device driver initialization,
2. Blank suppression off,
3. Use of the translation table specified during system initialization, (usually Ø29).

The card reader special function is requested using the following Assembly language sequence.

```
MOV #1,-(SP)        ; Push special function code
MOV #LNKBLK,-(SP)   ; Push LNKBLK address
EMT 12              ; Special function EMT
.
.
.
.
```

Special function codes not recognized by the system are ignored.

7.5 LS11 PRINTER SPECIAL FUNCTION (CODE 1)

The printer special function can only be specified for the Centronics printer. The printer special function code allows the user to specify whether the next line to be printed is permitted to use the elongated size or simply the normal character size. Because the elongated size doubles the width of the letters, no more than 66 characters can be printed on one line.

The special function block for the printer has the following format.

	1
	nnn

nnn equals Ø to disable character elongation.

nnn equals any number in the range 1 through 377_8 to enable elongation.

The actual code to command the line printer to elongate the line is 68_8.

PART 3

CHAPTER 8

EXAMPLE PROGRAMS

The two following example program listings illustrate methods for utilizing DOS/BATCH Monitor services. Note that the assembly language expansions of the programmed requests are used.

Example Program #1

```
                        ;PROGRAM WHICH TYPES A MESSAGE ON THE TELETYPE WHILE
                        ;ACCEPTING A MESSAGE FROM THE KEYBOARD.  PROGRAM REPEATS
1
2           000000 R0=%0
3           000001 R1=%1
4           000002 R2=%2
5           000003 R3=%3
6           000004 R4=%4
7           000005 R5=%5
8           000006 SP=%6
9           000007 PC=%7
10          000015 CR=15
11          000012 LF=12
12          000011 HT=11
13          000107 EROR=107
14 00000    012746 BEGIN:  MOV     #LNK1,-(SP)          ;INIT LNK1
            000312'
15 00004    104006         EMT     6
16 00006    012746         MOV     #LNK2,-(SP)          ;INIT LNK2
            000324'
17 00012    104006         EMT     6
18 00014    012746         MOV     #FIL2,-(SP)          ;OPEN FOR OUTPUT
            000356'
19 00020    012746         MOV     #LNK1,-(SP)          ;
            000312'
20 00024    104016         EMT     16
21 00026    012746         MOV     #FIL2,-(SP)          ;OPEN FOR INPUT
            000356'
22 00032    012746         MOV     #LNK2,-(SP)
            000324'
23 00036    104016         EMT     16
24 00040    012746         MOV     #MSG1,-(SP)          ;WRITE THE MESSAGE
            000370'
25 00044    012746         MOV     #LNK1,-(SP)
            000312'
26 00050    104002         EMT     2
27 00052    012700         MOV     #LIB1+6,R0           ;SET THE BUFFER POINTER
            000170'
28 00056    005020 LOOP1:  CLR     (R0)+                ;CLEAR THE ADDRESS AND INCREMENT
29 00060    020027         CMP     R0,#LIB1+80.         ;END OF BUFFER?
            000302'
30 00064    103774         BLO     LOOP1                ;NO, GO BACK AND CONTINUE CLEARIN
31 00066    012746         MOV     #LNK1,-(SP)          ;YES, CONTINUE
            000312'
```

```
32 00072 104001          EMT     1
33 00074 012746          MOV     #LIB1,-(SP)             ;NO,READ LNK2,LIB1
         000162'
34 00100 012746          MOV     #LNK2,-(SP)
         000324'
35 00104 104004          EMT     4
36 00106 012746          MOV     #LNK2,-(SP)             ;WAIT
         000324'
37 00112 104001          EMT     1
38 00114 132767          BITB    #EROR,LIB1+3            ;ANY ERRORS?
         000107
         000043
39 00122 001016          BNE     ERR3                    ;YES, GO TO THE ERROR #3 ADDRESS
40 00124 012746          MOV     #LNK1,-(SP)             ;NO, .CLOSE LNK1
         000312'
41 00130 104017          EMT     17
42 00132 012746          MOV     #LNK2,-(SP)             ;.CLOSE LNK2
         000324'
43 00136 104017          EMT     17
44 00140 016746          MOV     LNK1,-(SP)              ;.RLSE LNK1
         000146
45 00144 104007          EMT     7
46 00146 012746          MOV     #LNK2,-(SP)             ;.RLSE LNK2
         000324'
47 00152 104007          EMT     7
48 00154 000167          JMP     BEGIN
         177620
49 00160           ERR1:
50 00160           ERR2:
51 00160           ERR3:
52 00160 104060          EMT     60                      ;EXIT ON ANY ERROR
53 00162 000120 LIB1:    .WORD   80.                     ;MAX BYTE COUNT
54 00164    000          .BYTE   0,0                     ;FORMATTED ASCII
   00165    000
55 00166 000000          .WORD   0                       ;ACTUAL BYTE COUNT
56       000310'         .=.+80.                         ;RESERVE THE BUFFER SPACE
57 00310 000160'         .WORD   ERR1                    ;ERROR RETURN ADDRESS
58 00312 000000 LNK1:    .WORD   0                       ;POINTER
59 00314 016027          .RAD50  /DS1/                   ;LOGICAL NAME
60 00316    001          .BYTE   1,0                     ;UNIT 0
   00317    000
61 00320 042420          .RAD50  /KB/                    ;KEYBOARD
62 00322 000160'         .WORD   ERR2                    ;ERROR RETURN ADDRESS
63 00324 000000 LNK2:    .WORD   0
64 00326 016030          .RAD50  /DS2/                   ;
65 00330    001          .BYTE   1,0
   00331    000
66 00332 042420          .RAD50  /KB/                    ;KEYBOARD
67 00334 000000          .WORD   0                       ;GO TO FATAL ERROR MESSAGE
68 00336    002          .BYTE   2,0                     ;OPEN FOR OUTPUT
   00337    000
69 00340 000000 FIL1:    .WORD   0,0,0,0,0,0             ;NO NAME, EXT, UIC, OR PROTECT
   00342 000000
   00344 000000
   00346 000000
   00350 000000
70 00352 000000          .WORD   0                       ;GO TO FATAL ERROR
71 00354    004          .BYTE   4,0                     ;OPEN FOR INPUT
   00355    000
```

```
72 00356 000000 FIL2:   .WORD   0,0,0,0,0                ;NO NAME, EXT ,UIC, OR PROTECT
   00360 000000
   00362 000000
   00364 000000
   00366 000000
73 00370 000210 MSG1:   .WORD   210                      ;MAX BYTE COUN
74 00372     000        .BYTE   0,0                      ;FORMATTED ASCII
   00373     000
75 00374 000203         .WORD   MSGEND-MSG1-6            ;ACTUAL BYTE COUNT
76 00376     015        .BYTE   CR,LF,HT
   00377     012
   00400     011
77 00401     040        .ASCII  / SPEAK ROUGHLY TO YOUR LITTLE BOY /
   00402     123
   00403     120
   00404     105
   00405     101
   00406     113
   00407     040
   00410     122
   00411     117
   00412     125
   00413     107
   00414     110
   00415     114
   00416     131
   00417     040
   00420     124
   00421     117
   00422     040
   00423     131
   00424     117
   00425     125
   00426     122
   00427     040
   00430     114
   00431     111
   00432     124
   00433     124
   00434     114
   00435     105
   00436     040
   00437     102
   00440     117
   00441     131
   00442     040
78 00443     015        .BYTE   CR,LF,HT
   00444     012
   00445     011
79 00446     040        .ASCII  / AND BEAT HIM WHEN HE SNEEZES /
   00447     101
   00450     116
   00451     104
   00452     040
   00453     102
   00454     105
   00455     101
   00456     124
   00457     040
   00460     110
   00461     111
   00462     115
```

```
    00463    040
    00464    127
    00465    110
    00466    105
    00467    116
    00470    040
    00471    110
    00472    105
    00473    040
    00474    123
    00475    116
    00476    105
    00477    105
    00500    132
    00501    105
    00502    123
    00503    040
80  00504    015          .BYTE   CR,LF,HT
    00505    012
    00506    011
81  00507    040          .ASCII  / HE ONLY DOES IT TO ANNOY /
    00510    110
    00511    105
    00512    040
    00513    117
    00514    116
    00515    114
    00516    131
    00517    040
    00520    104
    00521    117
    00522    105
    00523    123
    00524    040
    00525    111
    00526    124
    00527    040
    00530    124
    00531    117
    00532    040
    00533    101
    00534    116
    00535    116
    00536    117
    00537    131
    00540    040
82  00541    000G         .BYTE   CR,LF,HT
    00542    011
83  00543    040          .ASCII  / BECAUSE HE KNOWS IT TEASES /
    00544    102
    00545    105
    00546    103
    00547    101
    00550    125
    00551    123
    00552    105
    00553    040
    00554    110
    00555    105
    00556    040
    00557    113
    00560    116
```

```
   00561     117
   00562     127
   00563     123
   00564     040
   00565     111
   00566     124
   00567     040
   00570     124
   00571     105
   00572     101
   00573     123
   00574     105
   00575     123
   00576     040
84 00577     015          .BYTE   CR,LF
   00600     012
85         000601'MSGEND=.
86                        .EVEN
87         000000'        .END    BEGIN

 BEGIN    000000R          CR     = 000015          CR.LF = ****** GX
 EROR   = 000107           ERR1     000160R         ERR2    000160R
 ERR3     000160R          FIL1     000340R         FIL2    000356R
 HT     = 000011           LF     = 000012          LIB1    000162R
 LNK1     000312R          LNK2     000324R         LOOP1   000056R
 MSGEND= 000601R           MSG1     000370R
 . ABS.  000000      000
         000602      001
 ERRORS DETECTED:  0
 FREE CORE:  9107. WORDS
 ,LP:<MON1.MAC
```

Example Program #2:

```
1                  ;PROGRAM TO SUPLICATE A PAPER TAPE
2                  ;USING TRAN-LEVEL REQUESTS
3                  ;
4          000000          R0=%0
5          000006          SP=%6
6          000007          PC=%7
7          000015          CR=15
8          000012          LF=12
9          000011          HT=11
10         000004          RD=04                   ;TRAN BLOCK FUNCTION CODE FOR .READ
11         000002          WR=02                   ;TRANBLOCK FUNCTION CODE FOR .WRITE
12         000107          G=107                   ;ASCII G
13         040000          EOD=40000                       ;TRANBLOCK FUNCTION/STATUS=EOD
14         000107          EROR=107
15 00000 012746 BEGIN:     MOV     #LNK1,-(SP)             ;.INIT LNK1
           000414'
16 00004 104006            EMT     6
17 00006 012746            MOV     #LNK2,-(SP)             ;.INIT LNK2
           000426'
18 00012 104006            EMT     6
19 00014 012746            MOV     #LNK3,-(SP)             ;.INIT LNK3
           000344'
20 00020 104006            EMT     6
21 00022 012746            MOV     #LNK4,-(SP)             ;.INIT LNK4
           000370'
```

```
22 00026 104006          EMT     6
23 00030 005067 START:   CLR     FLAG1                  ;ZERO END FLAG
         000210
24 00034 012767          MOV     #100.,BLK1+4           ;INITIALIZE BUFFER SIZE
         000144
         000342
25 00042 005067          CLR     BUF1+6                 ;INITIALIZE INPUT BUFFER
         000314
26 00046 005067          CLR     BUF1+1                 ;INITIALIZE INPUT BUFFER
         000303
27 00052 012746          MOV     #MSG1,-(SP)            ;.WRITE LNK3, MSG1
         000246'
28 00056 012746          MOV     #LNK3,-(SP)            ;
         000344'
29 00062 104002          EMT     2
30 00064 012746          MOV     #LNK3,-(SP)            ;.WAIT LNK3
         000344'
31 00070 104001          EMT     1
32 00072 012746          MOV     #BUF1,-(SP)            ;.READ LNK4, BUF1
         000354'
33 00076 012746          MOV     #LNK4,-(SP)
         000370'
34 00102 104004          EMT     4
35 00104 012746          MOV     #LNK4,-(SP)            ;.WAIT LNK4
         000370'
36 00110 104001          EMT     1
37 00112 132767          BITB    #EROR,BUF1+3
         000107
         000237
38 00120 001050          BNE     ERR6
39 00122 122767          CMPB    #G,BUF1+6              ;G?
         000107
         000232
40 00130 001337          BNE     START
41 00132 112767 LOOPR:   MOVB    #RD,BLK1+6             ;YES, SET UP READ
         000004
         000246
42 00140 012746          MOV     #BLK1,-(SP)            ;.TRAN LNK1, BLK1
         000400'
43 00144 012746          MOV     #LNK1,-(SP)
         000414'
44 00150 104010          EMT     10
45 00152 012746          MOV     #LNK1,-(SP)            ;.WAIT LNK1
         000414'
46 00156 104001          EMT     1
47 00160 032767          BIT     #EOD,BLK1+6            ;TEST FUNCTION FOR EOD
         040000
         000220
48 00166 001406          BEQ     LOOPW
49 00170 166767 ENDW:    SUB     BLK1+10,BLK1+4         ;RESET WORD COUNT TO FINAL
         000214
         000206
50                                                      ;         BUFFERS SIZE
51 00176 012767          MOV     #1,FLAG1               ;SET EOD FLAG
         000001
         000040
52 00204 112767 LOOPW:   MOVB    #WR,BLK1+6             ;SET UP WRITE
         000002
         000174
53 00212 012746          MOV     #BLK1,-(SP)            ;.TRAN LNKW,BLK1
         000400'
54 00216 012746          MOV     #LNK2,-(SP)
         000426'
```

```
55 00222 104010          EMT     10
56 00224 012746          MOV     #LNK2,-(SP)             ;,WAIT LNK2
         000426*
57 00230 104001          EMT     1
58 00232 005767          TST     FLAG1                   ;END OF DATA?
         000006
59 00236 001274          BNE     START                   ;YES,START OVER
60 00240 000734          BR      LOOPR                   ;NO, GET MORE
61 00242         ERR1:
62 00242         ERR2:
63 00242         ERR3:
64 00242         ERR4:
65 00242         ERR5:
66 00242         ERR6:
67 00242         ERR7:
68 00242 104060          EMT     60                      ;EXIT ON ANY ERROR
69 00244 000000 FLAG1:   .WORD   0                       ;1=>EOD RECEIVED ON READ
70 00246 000067 MSG1:    .WORD   55.
71 00250    000          .BYTE   0,0
   00251    000
72 00252 000067          .WORD   55.
73 00254    000G         .BYTE   CR,LF,HT
   00255    011
74 00256    114          .ASCII  /LOAD TAPE INTO READER/
   00257    117
   00260    101
   00261    104
   00262    040
   00263    124
   00264    101
   00265    120
   00266    105
   00267    040
   00270    111
   00271    116
   00272    124
   00273    117
   00274    040
   00275    122
   00276    105
   00277    101
   00300    104
   00301    105
   00302    122
75 00303    015          .BYTE   CR,LF,HT
   00304    012
   00305    011
76 00306    120          .ASCII  /PUSH   G, CR   WHEN READY/
   00307    125
   00310    123
   00311    110
   00312    040
   00313    040
   00314    040
   00315    107
   00316    054
   00317    040
   00320    103
   00321    122
   00322    040
   00323    040
   00324    040
   00325    127
```

```
   00326     110
   00327     105
   00330     116
   00331     040
   00332     122
   00333     105
   00334     101
   00335     104
   00336     131
77 00337     015          .BYTE   CR,LF
   00340     012
78                        .EVEN
79 00342 000242'          .WORD   ERR3
80 00344 000000 LNK3:     .WORD   0
81 00346 016027           .RAD50  /DS1/
82 00350     001          .BYTE   1,0
   00351     000
83 00352 042420           .RAD50  /KB/
84 00354 000004 BUF1:     .WORD   4
85 00356     000          .BYTE   0,0
   00357     000
86 00360 000004           .WORD   4
87       000366'          .=.+4
88                        .EVEN
89 00366 000242'          .WORD   ERR4
90 00370 000000 LNK4:     .WORD   0
91 00372 016027           .RAD50  /DS1/
92 00374     001          .BYTE   1,0
   00375     000
93 00376 042420           .RAD50  /KB/
94 00400 000000 BLK1:     .WORD   0
95 00402 000436'          .WORD BUF2
96 00404 000144           .WORD   100.
97 00406 000000           .WORD   0
98 00410 000000           .WORD   0
99 00412 000242'          .WORD   ERR3
100 0414 000000 LNK1:     .WORD   0
101 0416 016031           .RAD50  /DS3/
102 0420     001          .BYTE   1,0
    0421     000
103 0422 063320           .RAD50  /PR/
104 0424 000242'          .WORD   ERR2
105 0426 000000 LNK2:     .WORD   0
106 0430 016032           .RAD50  /DS4/
107 0432     001          .BYTE   1,0
    0433     000
108 0434 063200           .RAD50  /PP/
109 0436 000602'BUF2:     .=.+100.
110      000000'          .END    BEGIN
```

```
BEGIN   000000R         BLK1    000400R         BUF1    000354R
BUF2    000436R         CR    = 000015          CR,LF = ****** GX
ENDW    000170R         EOD   = 040000          EROR  = 000107
ERR1    000242R         ERR2    000242R         ERR3    000242R
ERR4    000242R         ERR5    000242R         ERR6    000242R
ERR7    000242R         FLAG1   000244R         G     = 000107
HT    = 000011          LF    = 000012          LNK1    000414R
LNK2    000426R         LNK3    000344R         LNK4    000370R
LOOPR   000132R         LOOPW   000204R         MSG1    000246R
RD    = 000004          START   000030R         WR    = 000002

. ABS.  000000      000
        000602      001
ERRORS DETECTED:  0
FREE CORE:  9091. WORDS
,LP:<MON2,MAC
```

PART 3

CHAPTER 9

SUMMARY OF MONITOR COMMANDS AND PROGRAMMED REQUESTS

9.1 SUMMARY OF MONITOR COMMANDS

Command	Usage
Commands to Allocate System Resources	
AS[SIGN]	Assign a physical device to a logical device name
Commands to Manipulate Core Images	
RU[N]	Load and begin a program
GE[T]	Load a program
DU[MP]	Write a specified core area onto a device as a core image
SA[VE]	Write a program onto a device in loader format
Commands to Start a Program	
BE[GIN]	Start execution of a program
CO[NTINUE]	Resume execution of a halted program
RE[START]	Restart execution of a previously operating program
Commands to Stop a Program	
ST[OP]	Halt the current program, including any I/O in progress
WA[IT]	Halt current program after finishing any I/O in progress
KI[LL]	Halt the current program, finish any I/O in progress, close all open files, and pass control back to the Monitor
Commands to Exchange Information with the System	
DA[TE]	Fetch/Specify date
TI[ME]	Fetch/Specify time
LO[GIN]	Enter User Identification Code
MO[DIFY]	Modify contents of memory location
FI[NISH]	Log off system

Command	Usage
Miscellaneous Commands	
EC[HO]	Disable/enable keyboard echo to user program
PR[INT]	Disable/enable terminal output from user program
EN[D]	End input from a device
OD[T]	Begin operation of On-Line Debugger (ODT)

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page

.ALLOC	Allocate a Contiguous File	.ALLOC #LNKBLK,#FILBLK,#N	MOV #N,-(SP)
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 15
				3-50

.APPND	Append to a Linked File	.APPND #LNKBLK,#FIRST,#SECOND	MOV #SECOND,-(SP)
			MOV #FIRST,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 22
				3-52

.BIN2D	Convert Binary to Decimal ASCII	.BIN2D #ADDR,#WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #3,-(SP)
			EMT 42
				3-53

.BIN2O	Convert Binary to Octal ASCII	.BIN2O #ADDR,#WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 42
				3-53

.BLOCK	Transfer a Block	.BLOCK #LNKBLK,#BLKBLK	MOV #BLKBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 11
				3-54

.CLOSE	Close a Dataset	.CLOSE #LNKBLK	MOV #LNKBLK,-(SP) EMT 17	3-55
.CORE	Obtain Core Size	.CORE	MOV #1ØØ,-(SP) EMT 41	3-56
.CSI1	CSI Interface - part 1	.CSI1 #CMDBUF	MOV #CMDBUF,-(SP) EMT 56	3-57
.CSI2	CSI Interface - part 2	.CSI2 #CSIBLK	MOV #CSIBLK,-(SP) EMT 57	3-58

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page

.CVTDT	Convert Binary Date or Time to ASCII character string	.CVTDT #CODE,#ADDR[,VALUE]
		VALUE is an optional argument specified with Codes 2 and 3 only.
			If Code = 3
			MOV VALUE+2,-(SP)
			If Code = 2 or 3
			MOV VALUE,-(SP)
			All codes
			MOV #ADDR,-(SP)
			MOV #CODE,-(SP)
			EMT 66
				3-61

.DATE	Obtain Date	.DATE	MOV #1Ø3,-(SP) EMT 41	3-62

.DELET	Delete a File	.DELET #LNKBLK,#FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 21
				3-63

.D2BIN	Convert Decimal ASCII to Binary	.D2BIN #ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 42
				3-64

.DUMP	Dump specified core locations to the line printer	.DUMP LOWADD,HIADD,CODE	MOV LOWADD,-(SP)
			MOV HIADD,-(SP)
			MOV CODE,-(SP)
			EMT 64
				3-95

.EXIT	Exit to Monitor	.EXIT	EMT 6Ø	3-64
.FLUSH	Bypasses lines in the batch stream.	.FLUSH CODE	EMT 67	3-97
.GTCIL	Get Disk address of Core Image library	.GTCIL	MOV #111,-(SP) EMT 41	3-65
.GTCLK	Obtain system clock information	.GTCLK	MOV #113,-(SP) EMT 41	3-65
.GTOVF	Obtain and set the overlay flag	.GTOVF	MOV #114,-(SP) EMT 41	3-66

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.GTPLA	Get Program Low Address	.GTPLA	CLR -(SP) MOV #5,-(SP) EMT 41	3-66
.GTRDV	Obtain RUN device information	.GTRDV	CLR -(SP) CLR -(SP) MOV #112,-(SP) EMT 41	3-67
.GTSTK	Get the Stack Base Address	.GTSTK	CLR -(SP) MOV #4,-(SP) EMT 41	3-67
.GTUIC	Get Current UIC	.GTUIC	MOV #1Ø5,-(SP) EMT 41	3-68
.INIT	Initialize a Dataset	.INIT #LNKBLK	MOV #LNKBLK,-(SP) EMT 6	3-68

.LOOK	Directory Search	.LOOK #LNKBLK,#FILBLK[,1]
		,1 is an optional argument
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
			or when optional argument is specified:
			MOV #FILBLK,-(SP)
			CLR -(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
				3-69

.MONF	Obtain Full Monitor Size	.MONF	MOV #1Ø2,-(SP) EMT 41	3-71
.MONR	Obtain Size of Resident Monitor	.MONR	MOV #1Ø1,-(SP) EMT 41	3-72

.OPEN	Open a Dataset	.OPEN #LNKBLK,#FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
				3-72

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page

.OPENx	Open a Dataset	.OPENx #LNKBLK,R	MOV #CODE,-2(R)
			MOV R,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
			
			CODE = 1 for .OPENU
			2 for .OPENO
			3 for .OPENE
			4 for .OPENI
			13 for .OPENC
				3-72

.O2BIN	Convert Octal ASCII to Binary	.O2BIN #ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 42
				3-76

.RADPK	Radix-5Ø ASCII Pack	.RADPK #ADDR	MOV #ADDR,-(SP) CLR -(SP) EMT 42	3-76

.RADUP	Radix-5Ø ASCII Unpack	.RADUP #ADDR,#WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #1,-(SP)
			EMT 42
				3-79

.READ	Read from Device	.READ #LNKBLK,#BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 4
				3-80

.RECRD	Read or Write a Specified Record in a File	.RECRD #LNKBLK,#RECBLK	MOV #RECBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 25
				3-81

.RENAM	Rename a File	.RENAM #LNKBLK,#OLDNAM,#NEWNAM	MOV #NEWNAM,-(SP)
			MOV #OLDNAM,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2Ø
				3-82

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.RLSE	Release a Dataset	.RLSE #LNKBLK	MOV #LNKBLK,-(SP) EMT 7	3-83

.RSTRT	Set REstart address	.RSTRT #ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 41
				3-84

.RUN	Load a program or Overlay	.RUN #RUNBLK	MOV #RUNBLK,-(SP) EMT 65	3-84

.SPEC	Special Function	.SPEC #LNKBLK,#SPCARG	MOV #SPCARG,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 12
				3-86

.STAT	Obtain Device Status	.STAT #LNKBLK	MOV #LNKBLK,-(SP) EMT 13	3-87

.STFPU	Initialize the Floating Point exception vector (11/45)	.STFPU #PSW,#ADDR	MOV #ADDR,-(SP)
			MOV #PSW,-(SP)
			MOV #3,-(SP)
			EMT 41
				3-88

.STPLA	Set Program Low Address	.STPLA #ADDR	MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 41
				3-89

.STSTK	Set the Stack Base Address	.STSTK #ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 41
				3-89

.SYSDV	Obtain System Device Name	.SYSDV	MOV #1Ø6,-(SP) EMT 41	3-90
.TIME	Obtain Time of Day	.TIME	MOV #1Ø4,-(SP) EMT 41	3-91

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page

.TRAN	Transfer Absolute Block	.TRAN #LNKBLK,#TRNBLK	MOV #TRNBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 1Ø
				3-91

.TRAP	Set TRAP Vector	.TRAP #STATUS,#ADDR	MOV #ADDR,-(SP)
			MOV #STATUS,-(SP)
			MOV #1,-(SP)
			EMT 41
				3-93

.WAIT	Wait for Completion	.WAIT #LNKBLK	MOV #LNKBLK,-(SP) EMT 1	3-93

.WAITR	Wait for Completion; Return to ADDR	.WAITR #LNKBLK,#ADDR	MOV #ADDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT Ø
				3-94

.WRITE	Write on a Device	.WRITE #LNKBLK,#BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2
				3-95